64 FLOPPY

Supplemento a Special Program
Anno IV - N. 30

IL KIT DEL PROGRAMMATORE



SUL DISCHETTO:

- TAVOLETTA GRAFICA
- EDITOR SPRITES
- EDITOR CARATTERI
- COPIA PROGRAMMI
- COPIA DISCHI
- ALLINEA TESTINE
- ASSEMBLATORE

CONTIENE UN POSTER DI UTILITY



Benvenuti nel mondo dei programmatori.
Avete fatto una buona scelta! In questa nuova rivista infatti troverete i principi della programmazione.
La maggior parte della rivista è dedicata a un assemblatore che vi permetterà di ''trafficare'' nella maniera migliore e più semplice.
Il primo ''tool'' di lavoro è comunque una tavoletta grafica che vi darà la possibilità di disegnare a vostro piacimento. Poi troverete un Editor di sprite e uno di caratteri. Quindi non potevamo non darvi due copiatori, uno di programmi (files) e uno di dischi interi.
Infine oltre all'Assemblatore di cui vi abbiamo già detto, c'è un comodissimo allineatore di testina del vostro disk drive che vi farà risparmiare tempo e soldi.

SOMMARIO

Istruzioni Tavoletta Grafica	**pag. 3**
Editor Sprites	**pag. 3**
Editor Caratteri	**pag. 4**
Copia Programmi	**pag. 5**
Copia Dischi	**pag. 5**
Allinea Testine	**pag. 5**
Assemblatore	**pag. 6**
Programmer - Poster	**pag. 16**

Per caricare digitare
LOAD "*", 8,1 e RETURN

Edizione SIPE s.r.l. - Via Ausonio, 26 - 20123 Milano - Direttore Resp. Elvio Fantini
Suppl. a Special Program n. 30 - Registr. presso il Trib. di Milano N. 278 del 2-6-84 - Sped. in Abb. Post. Gruppo III/70%
Distrib.: MEPE - Viale Famagosta, 75 - Milano - Stampa: Arti Grafiche Franco Battaia & C. - Rozzano (MI)

TAVOLETTA GRAFICA



KOALA

Usa il joystick in porta 2 oppure:
F1 = ↑ F3 = ↓ F5 = ← F7 = →
Spazio = Fuoco

		HEX	DEC
Inizio	pagina grafica (bit map)	6ØØØ	24576
Fine		7F3F	32575
Inizio	colore N. 1	8328	33576
Fine		87ØF	34575
Inizio	colore N. 2	7F4Ø	32576
Fine		8327	33575
Valore da Trasferire In DØ21	(col. fondo)	871Ø	34576

FILE $6000 – $8710

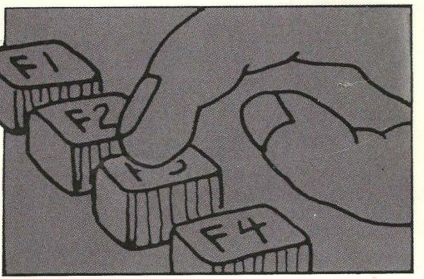
EDITOR SPRITES

L per caricare un file di dati
S per salvare un file di dati
D per stampare sul video i dati delle sprite selezionate.
+ per sovrapporre alla sprite visualizzata un'altra sprite
HOME per portare il cursore in alto a sinistra
CLR per cancellare la sprite visualizzata
P per visualizzare una sprite precedente
* per accendere il punto sotto il cursore
F per "riempire" la sprite visualizzata
G per puntare ad un'altra sprite
X per abortire "sprite maker"
C per copiare una sprite
N per puntare alla sprite successiva
M per passare in "multicolor mode"
H per passare in "High-Resolution Mode"
<SPACE> per cancellare un punto puntato
Cursori per puntare un punto
F1 per cambiare il colore di fondo
F3 per cambiare il colore N. 1
F5 per cambiare il colore N. 2
F7 per cambiare il colore principale

Dati Tecnici	HEX	DEC
Inizio File	3200	12800
Fine File	3FFF	16383



EDITOR CARATTERI



E ERASE per cancellare un blocco di caratteri
< SCRSIN scroll finestra a sinistra
> SCRDES scroll finestra a destra
O SCRSU scroll finestra in alto
L SCRGIU scroll finestra in basso
X MAPPA per variare la mappa caratteri
(tasto CLR-Home + Shift) clear pulisce finestra
F FILL riempie finestra
C COPY RAM copia un blocco di caratteri
Y COPY ROM copia un blocco di caratteri prelevandoli dal set standard
N NEXT punta al carattere successivo
P PREVIOUS punta al carattere precedente

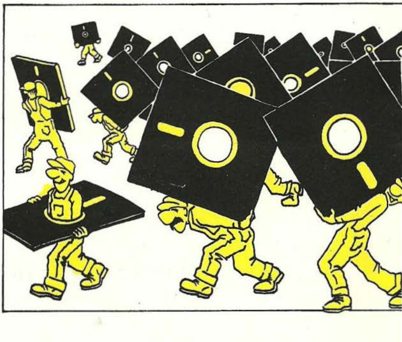
G GOTO punta al carattere che viene scelto
↑ HOME il cursore nella finestra va in alto a sin.
M MULTI passa in multicolor mode
H HIRES passa in alta risoluzione
Z MEMORIZZA memorizza un carattere
V RISCRIVE riscrive un carattere
Q QUIT torna al basic lasciando lo split del video
1,2&3 COLORI cambiano i colori: caratteri, multi 1, multi 2
S SAVE salva un set su disco
A LOAD carica un set su disco
F8 SCRIVE scrive un carattere editato
F1 MATRICE legge un carattere da editare

COPIA DISCHI



Questo programma serve per effettuare il back-up completo di un disco. La sua originalità sta nel fatto che il programma chiede ed esegue tutto ciò che noi gli vogliamo far fare. Ad esempio si può indicare e copiare solo le tracce che noi desideriamo guadagnando in tempo e spazio sul dischetto.

COPIA PROGRAMMI



È forse uno dei copiatori più utili che i programmatori del 64 abbiano pensato. Infatti in un solo programma, senza bisogno di trafficare con dischetti e programmi vari, si può copiare un programma, vedere la directory del disco, inserire i comandi di lavoro, analizzare lo status del dischetto e uscire dal programma.

F1 copia il file
F3 dà la directory
F5 comandi disco
F7 status disco
F8 esce dal programma
SPAZIO torna al menu

ALLINEAMENTO TESTINA



Un semplice e breve programma che vi fa risparmiare un sacco di tempo e permette di avere sempre la testina del drive nella giusta posizione. Dopo aver caricato il programma bisogna attendere affinché la testina si posizioni correttamente e il tempo di accesso al disco sia inferiore a 9. Comunque il programma vi indica chiaramente sul video le operazioni che il drive sta svolgendo.

ASSEMBLATORE

Questo manuale descrive il linguaggio assembler e le procedure per assemblare programmi per il Commodore 64 che usino uno dei processori della serie 6500. In commercio esistono numerosi assembler per lo sviluppo di programmi con la serie 6500, ed ognuno ha delle modalità di utilizzo leggermente diverse, pur mantenendo le medesime linee generali. La serie di processori 6500 comprende tutti quelli dal 6502 al 6515 (il set di istruzioni è identico).
Il processo di traduzione di un programma dalla forma mnemonica o simbolica al vero codice macchina è chiamato assemblaggio o compilazione (d'ora in poi questi termini verranno usati indifferentemente), e un programma che effettui questa traduzione è detto assembler. Noi chiameremo CODICE SORGENTE la forma simbolica di un programma, CODICE OGGETTO il vero codice macchina costituente il programma stesso. Generalmente un comando in linguaggio assembler viene tradotto in una singola istruzione in codice macchina; questo distingue un assembler da un compilatore ad alto livello, in quanto quest'ultimo può generare più istruzioni da una sola. Un assembler che opera su di un computer diverso da quello per cui il codice è generato è chiamato cross-assembler. Spesso l'uso di un cross-assembler si rende necessario perché un computer non ha sufficiente potenza per gestire un buon assembler, ma nel caso del Commodore 64 non ci sono tali problemi: con un floppy disk ed una stampante il sistema è adatto allo sviluppo di software.
Solitamente i computer digitali usano la numerazione binaria per rappresentare dati ed istruzioni, in quanto possono gestire solo 'uno' e 'zero', corrispondenti agli stati acceso e spento. Per gli uomini, d'altra parte, è molto difficile lavorare con la numerazione binaria, quindi si usano delle numerazioni più convenienti, in base 8 (ottale), in base 10 (decimale) ed in base 16 (esadecimale). Due rappresentazioni dell'operazione di caricamento di un valore nell'accumulatore sono:
10101001 (binario)
A9 (esadecimale)
Un'istruzione per porre il valore 21 (decimale) nell'accumulatore è:
A9 15 (esadecimale)
La rappresentazione numerica delle istruzioni è comunque piuttosto difficile da ricordare, per questo sono state inserite delle rappresentazioni simboliche con le quali è molto più semplice lavorare. Per esempio l'istruzione precedente potrebbe essere scritta così:
LDA #21
In questo esempio LDA è la rappresentazione simbolica per A9, e significa 'carica l'accumulatore' (in inglese LoaD the Accumulator). Un assembler può trasformare la scrittura simbolica LDA nella forma numerica A9.
Ogni istruzione in codice macchina ha un proprio nome simbolico chiamato CODIFICA. La codifica per 'salva l'accumulatore' è STA. La codifica per 'trasferisci l'accumulatore al registro X' è TAX. Le 56 codifiche proprie della serie di processori 6500 sono elencate nella Appendice IV. Un'istruzione in linguaggio assembler consiste in una codifica, seguita talvolta da un operando, che specifica il dato su cui agisce l'operazione.
Un'etichetta è un 'nome' per una linea di programma; può essere apposta per creare un riferimento ad uso di altre istruzioni, come si può vedere:
L2 LDA #12
L'etichetta è L2, la codifica è LDA e l'operando è #12. Almeno uno spazio deve separare le tre parti (campi) di un'istruzione, e gli spazi addizionali che possono essere aggiunti per facilitare la lettura da parte del programmatore vengono ignorati dall'assembler.
Le istruzioni per la serie di processori 6500 hanno al massimo un operando, ed alcune ne sono del tutto prive. In questi casi l'operazione che va eseguita è totalmente specificata dalla codifica, come ad es. CLC (azzera il flag di carry o riporto).
Programmare in linguaggio assembler richiede l'apprendimento del set di istruzioni (codifiche), delle convenzioni di indirizzamento per il riferimento ai dati, delle strutture dei dati all'interno del processore, e naturalmente della struttura dei programmi in linguaggio assembler.
Parte terza. 'Creare e modificare file sorgenti in linguaggio assembler', è composta dalle sezioni 5-6, e descrive come creare e modificare un file sorgente in linguaggio assembler. La sezione 5 contiene le istruzioni per caricare un programma di supporto, il programma WEDGE. Questo programma fornisce all'utente dei comandi addizionali per utilizzare il disco, caricare ed eseguire programmi. La sezione 6 con-

tiene le istruzioni operative per caricare ed eseguire il programma EDITOR 64, che permette di creare e modificare file sorgenti in assembler.

Parte quarta. 'Assemblare e controllare un programma', è composta dalle sezioni 7-9, e contiene informazioni sul programma che permette all'utente di assemblare, controllare ed eventualmente correggere un programma oggetto. La sezione 7 descrive le operazioni del programma assembler; la sezione 8 descrive il programma che deve essere usato per caricare un programma oggetto in memoria; la sezione 9 descrive il programma che permette di esaminare la memoria per correggere gli errori (monitor).

Parte quinta. 'Appendici', contiene quelle tabelle che possono essere usate come riferimento ad altre sezioni. Fornisce anche un riferimento rapido ai comandi disponibili quando si eseguono alcuni programmi.

In questo manuale ci sono alcune convenzioni atte a rendere più chiare le spiegazioni. Eccone l'elenco:

( )

Sono usate per indicare che l'argomento in esse racchiuso è opzionale. Le sole eccezioni a questa regola sono in quelle sezioni dove sono spiegati l'indirizzamento indiretto indicizzato e l'indirizzamento indicizzato indiretto. In questi casi le parentesi sono necessarie.

Etichetta

È usato per indicare un riferimento ad un'etichetta nel codice sorgente di un programma. La vera etichetta usata è determinata dal programmatore.

Codifica

È usato per indicare una delle istruzioni del 6502 elencate nella Appendice IV.

Operando

È usato per indicare l'operando o l'argomento di un'istruzione.

Nomefile

È usato per specificare il nome di un file su disco. Il vero nome è determinato dall'utente.

Nomefile*

È usato per indicare il nome di più file (ovvero un nome che finisca con '*').

Variabile min.

Generalmente le variabili in minuscolo indicano che è compito dell'utente specificare quel dato.

Nome Maiuscolo

Generalmente i nomi in maiuscolo indicano l'input da battere.

1.0 Convenzioni sul formato delle istruzioni

Le istruzioni per l'assembler del Commodore 64 si dividono in due tipi fondamentali:

* istruzioni di linguaggio macchina
* comandi per l'assembler

Le istruzioni di linguaggio macchina sono le 56 operazioni implementate sulla serie di processori 6500. Il formato delle istruzioni è:

(etichetta) codifica (operando) (commento)

I campi chiusi tra parentesi sono opzionali, ovvero possono essere usati oppure omessi a discrezione dell'utente.

Etichette e commenti sono sempre opzionali, e molte codifiche come RTS (ritorna da una subroutine) non richiedono operando. Una linea può anche contenere solo un'etichetta o un commento.

Una tipica istruzione che mostra tutti e quattro i campi è:

LOOP LDA BETA, X ;
PRENDI BETA INDICIZZATO DA X

Un campo è definito come una stringa di caratteri separati da uno spazio. Un'etichetta è una stringa alfanumerica da uno a sei caratteri, il primo dei quali deve essere una lettera. Un'etichetta non può essere nessuna delle 56 codifiche, né uno dei caratteri speciali usati dall'assembler come riferimento a:

Accumulatore (A)
Puntatore dello stack (S)
Stato del processore (P)
Registri indice (X e Y)

Un'etichetta può iniziare in qualsiasi colonna sempreché sia il primo campo di un'istruzione. Le etichette sono usate nelle istruzioni come identificatori di una linea, nelle dichiarazioni di dati come riferimento per gli operandi.

L'operando di un'istruzione specifica o un indirizzo o un valore.

Un indirizzo può essere calcolato tramite il valore di un'espressione, e l'assembler permette una notevole flessibilità nella formazione di espressioni. Un'espressione in linguaggio assembler consiste in una stringa di nomi e costanti (che d'ora in poi chiameremo anche simboli) separati dai segni operativi +, —, *, / (addizione, sottrazione, moltiplicazione e divisione).

Le espressioni sono valutate dall'assembler per calcolare l'indirizzo dell'operando. Le espressioni sono calcolate da destra a sinistra senza alcuna precedenza tra i segni operativi e senza considerare le parentesi.

Ricordate che le espressioni vengono calcolate al momento della compilazione e non mentre il programma viene eseguito.

Qualsiasi stringa di caratteri che segua il campo degli operandi è considerata un commento. I commenti sono listati nel codice sorgente, ma vengono ignorati dall'assembler. Se il primo carattere di una riga è un punto e virgola tutto ciò che segue è considerato un commento. In un'istruzione che non richiede operando un commento può seguire la codifica, separato da questa mediante almeno uno spazio. L'appendice V mostra il listato del codice sorgente di un semplice programma. Sono inclusi vari esempi di formato delle istruzioni.

1.1 SIMBOLICO

Forse il più comune modo di indirizzamento degli operandi è la forma simbolica, come in:

LDA BETA ; METTI IL VALORE BETA NELL'ACCUMULATORE

In questo esempio BETA è un'etichetta che si riferisce ad un byte nella memoria contenente il valore da caricare nell'accumulatore. BETA è un'etichetta o un indirizzo al quale il valore si trova. Similmente nell'istruzione:

LDA ALFA + BETA

l'indirizzo ALFA + BETA è calcolato dall'assembler, ed il valore all'indirizzo di memoria è posto nell'accumulatore.

La memoria associata alla serie di processori 6500 è segmentata in pagine lunghe 256 byte ciascuna. La prima pagina, pagina zero, è trattata dall'assembler e dal processore in modo differente rispetto alle altre, per l'ottimizzazione dello spazio in memoria. Molte delle istruzioni hanno codici operativi alternativi se l'indirizzo dell'operando è nella pagina zero. In questi casi l'indirizzo è lungo un solo byte invece dei normali due. Per esempio:

LDA BETA

Se BETA è posto al byte 4B nella pagina zero, allora il codice generato è A5 B4. Questo è chiamato indirizzamento in pagina zero. Se BETA è al byte 01 3C nella pagina di memoria uno, il codice generato è AD 3C 01. Questo è un esempio di indirizzamento 'assoluto'.

Così, per ottimizzare la quantità di memoria utilizzata ed il tempo di esecuzione, un buon programmatore dovrebbe progettare programmi i cui dati risiedano nella pagina zero, quando ciò è possibile.

Bisogna invece evitare di assemblare programmi nella pagina zero, poiché potrebbero sorgere alcuni problemi.

Ricorda, l'assembler decide che formato usare in base al calcolo dell'indirizzo dell'operando.

1.2 Costanti

I valori costanti in linguaggio assembler possono avere più formati. Se una costante non è decimale, si usa un carattere di prefisso per specificarne il tipo:

$ specifica che il valore che segue è esadecimale

@ specifica che il valore che segue è ottale

% specifica che il valore che segue è binario

' specifica che bisogna considerare il valore ASCII del carattere che segue.

L'assenza di un prefisso indica che il valore è decimale.

Nel comando:

LDA BETA + 5

il numero decimale 5 è sommato a BETA per calcolare l'indirizzo. Similmente:

LDA BETA + $5F

fa sì che il valore esadecimale 5F sia sommato a BETA per calcolare l'indirizzo.

Il modo di indirizzamento immediato è indicato dal simbolo '#' seguito da una costante. Per esempio:

LDA #2

fa sì che il valore decimale 2 sia posto nell'accumulatore. Similmente:

LDA #'G

carica il valore ASCII di 'G' nell'accumulatore. Giacché l'accumulatore è lungo un solo byte, il valore caricato deve essere compreso tra 0 e 255 decimali.

Il modo di indirizzamento immediato genera due o tre byte di codice (dipende se l'indirizzo si trova nella pagina zero oppure no), ovvero la codifica ed il valore da usare come operando. Nota che le costanti possono essere usate sia nelle espressioni di indirizzo sia come valori nel modo di indirizzamento immediato. Possono anche essere usate per inizializzare delle locazioni di memoria, come è spiegato più avanti.

1.3 Salti relativi

Sono disponibili otto istruzioni di salto condizionale. In questo esempio:

BEQ START ; SE UGUALE VAI A START

Se i valori confrontati sono uguali, l'esecuzione del programma salta alla linea avente etichetta START. L'indirizzo di salto relativo è un numero da —128 a +127, che viene sommato (algebricamente) al contatore di programma durante l'esecuzione. Quando viene eseguita questa addizione il contatore di programma punta all'istruzione successiva all'istruzione di salto. Il salto parte dalla locazione dell'istruzione successiva. Durante la compi-

lazione viene evidenziato un errore se il salto è fuori dai limiti consentiti (ovvero se l'istruzione a cui il programma dovrebbe saltare è troppo lontana dall'istruzione di salto).
L'indirizzamento relativo è usato esclusivamente nelle istruzioni di salto.

1.4 Istruzioni con operando implicito
Venticinque istruzioni come TAX (trasferisci l'accumulatore al registro X), non richiedono alcun operando, occupando, in questo modo, un solo byte. In queste istruzioni infatti l'operando è implicito nella codifica.
Quattro istruzioni (ASL, LSR, ROL e ROR) sono particolari in quanto in esse l'accumulatore, A, può essere usato come operando. In questo speciale caso esse sono trattate come indirizzamenti in modo implicito, e viene generata una sola codifica.

1.5 Indirizzamento indiretto indicizzato
In questo modo, l'indirizzo dell'operando è calcolato sommando il registro X (l'indice) all'argomento dell'operando (nell'esempio seguente, BETA). Il valore risultante è l'indirizzo delle locazioni di pagina zero che contengono l'effettivo indirizzo dell'operando. In questo esempio:
LDA (BETA, X)
le parentesi attorno all'operando indicano il modo indiretto. Nell'esempio sopra il valore del registro indice X è sommato a BETA; la somma si riferisce in ogni caso ad una locazione della pagina zero, in quanto durante l'esecuzione il byte alto viene ignorato. I due byte a partire da quella locazione sono assunti come indirizzo dell'operando, nel formato byte basso-byte alto.
Per capire meglio, considera ciò che segue:
BETA contiene $12
X contiene $4
le locazioni $0017 e $0016 contengono $01 e $25
La locazione $0125 contiene $37
BETA + X è uguale a $16, l'indirizzo alla locazione $16 è $0125. Il valore a $0125 è $37, quindi l'istruzione LDA (BETA, X) carica il valore $37 nell'accumulatore.
L'indirizzamento indiretto indicizzato è usato spesso per accedere ad una tabella di vettori di indirizzo posta nella pagina zero.
La figura seguente illustra questa forma di indirizzamento.

1.6 Indirizzamento indicizzato indiretto
Un altro modo di indirizzamento usa il registro Y ed è illustrato da:
LDA (GAMMA, Y)
In questo caso GAMMA si riferisce alla locazione di pagina zero nella quale si trova un indirizzo. Il valore di Y è sommato a quell'indirizzo per calcolare l'effettivo indirizzo dell'operando. Considera ciò che segue:
GAMMA contiene $38
Y contiene $7
Le locazioni $0039 e $0038 contengono $00 e $54
La locazione $005B contiene $26
L'indirizzo a $38 è $0054, ad esso viene sommato $7, ottenendo così un indirizzo effettivo uguale a $005B. Il valore a $005B è $26, che viene caricato nell'accumulatore.
Nel modo indicizzato indiretto, l'indice X è sommato all'operando prima dell'indirizzamento; nel modo indiretto indicizzato, al contrario, prima è effettuato l'indirizzamento, poi l'indice Y è sommato per calcolare l'indirizzo effettivo. Il modo indiretto è sempre indicizzato tranne che nell'istruzione JMP, che permette un indirizzamento indiretto assoluto, come esemplificato da JMP (DELTA), che causa un salto all'indirizzo contenuto nelle locazioni DELTA e DELTA + 1.

2.0 Comandi per l'assembler
Sono disponibili undici comandi riservati all'assembler, usati per riservare aree di memoria, per fornire informazioni direttamente all'assembler e per 'dargli ordini', se così si può dire. Nove di essi hanno nomi simbolici il cui primo carattere è un punto. Il decimo, il segno di uguale (=), stabilisce un'identità fra un'espressione ed un valore. L'undicesimo, l'asterisco (*), rappresenta il valore del contatore della locazione corrente. Esempi di identità sono: ROSSO = 5, BLU = $FF e * = 200. Un elenco dei comandi è stampato qui sotto (il loro uso è spiegato nella sezione seguente).

```
.BYTE   .WORD   .DBYTE   .PAGE   .SKIP
.OPT         .END         .FILE        .LIB
=
*
```

Le etichette e tutte le espressioni che non siano comandi non possono iniziare con un punto.
Esempi di comandi all'assembler possono essere visti nel semplice programma assembler listato nell'Appendice V.
Se lo si desidera, si possono abbreviare i co-

mandi che cominciano con un punto, battendo solo il punto seguito dai primi tre caratteri (ad es. '.DBY' invece di '.DBYTE').

.BYTE è usato per riservare un byte di memoria e per immagazzinare in esso un valore. Il comando può contenere operandi multipli, che porranno più valori in byte consecutivi. Stringhe ASCII possono essere generate racchiudendole tra virgolette semplici ('). Va ricordato che esiste un limite di 40 caratteri per ogni comando .BYTE.

```
QUI     .BYTE 2
QUO     .BYTE 1,$ F, @3, %101,7
QUA     .BYTE 'ABCDEFGH'
```

Nota che i numeri possono essere scritti nella forma più comoda; in generale ogni espressione, la cui soluzione sia un numero compreso tra 0 e 255 (un byte), può essere usata in questi comandi. Se vuoi includere una virgoletta in una stringa ASCII, inserisci due virgolette nella stringa stessa.
Per esempio:
.BYTE 'MOLLA L' 'OSSO'
può essere usato per memorizzare:
MOLLA L'OSSO
Bisogna notare che l'uso di operazioni aritmetiche nel comando .BYTE non è supportato in questa versione del programma.

.WORD è usato per riservare e caricare due byte di dati per volta. Ogni espressione valida, tranne le stringhe ASCII, può essere usata nel campo dell'operando. Per esempio:

```
QUI .WORD 2
QUO .WORD 1, $FF03, @ 3
QUA .WORD QUI,QUO
```

L'uso più comune di .WORD è per generare indirizzi, come mostrato nell'ultimo esempio, che immagazzina gli indirizzi a 16 bit QUI e QUO. Nella serie 6500 gli indirizzi vengono posti in memoria nell'ordine byte basso-byte alto, dunque .WORD genera valori in quest'ordine.
La parte esadecimale nell'esempio verrebbe immagazzinata come 03,FF. Se non vuoi questo ordine, usa .DBYTE al posto di .WORD.

.DBYTE è praticamente identico a .WORD, tranne per il fatto che i byte sono immagazzinati nell'ordine byte alto-byte basso. Per esempio:
.DBYTE $FF03
genera FF,03. Per questo, i campi generati da .DBYTE non possono assolutamente essere usati come indirizzi.

Uguale (=) è il comando di uguaglianza, ed è usato per riservare locazioni di memoria, per resettare il contatore di programma (*), o per assegnare un valore ad un'espressione.

```
QUI * = 1+1 riserva un byte
QUO * = * + 2 riserva due byte
* = $200 fissa il cont. di programma
NB = B assegna un valore
MB = NB + %101 assegna un valore
```

Il comando ' = ' è molto potente e può essere usato in una grande varietà di occasioni.

Asterisco (*) è usato per modificare il valore del contatore di programma. Per creare un programma oggetto che venga assemblato a partire da un qualsiasi indirizzo maggiore di zero, deve essere usato il comando '*'. Per esempio '* =$200', fa partire la compilazione all'indirizzo $200.
Le espressioni non devono contenere riferimenti a nomi definiti più avanti, o saranno segnalate come errori. Ad esempio:
* = C + D — E + F
viene considerata valida se C,D,E ed F sono state tutte definite, è invece segnalata come errore se una delle variabili è definita più avanti nel programma, o se non è definita affatto. Ricorda inoltre che l'espressione è calcolata sempre da sinistra verso destra, senza alcuna priorità.

.PAGE è usato per provocare un salto immediato all'inizio di pagina nel listato, e può essere usato per generare o per resettare il titolo stampato sul listato.

```
.PAGE Questo è un titolo
.PAGE
.PAGE Nuovo titolo
```

Se viene definito un titolo, esso viene stampato in cima ad ogni pagina fino a che non viene cancellato o sostituito con un altro. Un titolo può essere cancellato con:
.PAGE ' '

.SKIP è usato per generare linee vuote in un listato. Il comando non appare, ma la sua posizione può essere trovata in un listato. Il comando, pur non essendo stampato, è considerato come un inserimento valido dal listato, ed il numero sulla sinistra del listato stesso salta di due quando la linea seguente viene stampata.

```
.SKIP 2 salta due linee vuote
.SKIP 3*2—1 salta cinque linee vuote
.SKIP salta una linea
```

.OPT è il comando più potente ed è usato per controllare la creazione dei campi di output, dei listati e delle espansioni delle stringhe ASCII nei comandi .BYTE. Le opzioni dispo-

nibili sono: ERRORS, NOERRORS, LIST, NOLIST, GENERATE, NOGENERATE.
.OPT ERRORS, LIST, GENERATE
.OPT NOE, NOL, NOG
è anche valido:
.OPT LIST, ERR
I valori per difetto sono:
.OPT LIST, ERR, NOGEN
Ecco qui la descrizione di ogni opzione:

ERRORS-NOERRORS:
Usato per controllare la creazione di un file separato per gli errori. Il file di errore contiene la linea sorgente sbagliata ed il messaggio di errore. Questa possibilità è di grande aiuto a quegli utenti che lavorano in time-sharing i quali hanno limitate capacità di stampa. Il file di errore può essere attivato ed esaminato fino a che tutti gli errori sono stati corretti. Questo file di listato può poi essere esaminato. Un'altra possibilità è di lanciare la compilazione con:
.OPT ERROR, NOLIST
Fino a che tutti gli errori sono stati corretti, e quindi rilanciare con:
.OPT NOERRORS, LIST

LIST-NOLIST:
Usato per controllare la generazione dei file di listato contenente il programma sorgente, le segnalazioni di errore o pericolo, la generazione del codice, la tabella dei simboli ed il conteggio delle istruzioni (se abilitato).

GENERATE-NOGENERATE:
Usato per controllare la stampa delle stringhe ASCII nel comando .BYTE. I primi due caratteri vengono sempre stampati, mentre i caratteri seguenti vengono stampati (normalmente due byte per linea) solo se si usa GENERATE.

.END dovrebbe essere l'ultimo comando in un file ed è usato per segnalare la fine fisica del file stesso. Il suo uso è opzionale, ma lo si raccomanda caldamente per la documentazione del programma.

.LIB permette all'utente di inserire codice sorgente proveniente da un altro file nell'assembler. Quando l'assembler incontra questo comando smette momentaneamente di leggere il codice sorgente dal file corrente ed inizia a leggere dal file nominato in .LIB. La lettura del file corrente ricomincia quando l'assembler incontra la fine del file libreria (EOF) oppure il comando .END. Il file contenente la .LIB può a sua volta contenere comandi per l'assembler che controllino le funzioni di listing, etc...

.FIL può essere usato per concatenare un altro file al corrente durante la compilazione. Un file libreria non può contenere un altro .LIB, ma può contenere un comando .FIL. Un comando .FIL conclude la compilazione dei file in cui è contenuto e trasferisce la lettura del codice sorgente ad un altro file nominato nel campo operando. Non ci sono restrizioni al numero di file che possono essere concatenati con il comando .FIL. Bisogna fare attenzione quando si usa questo comando a non creare concatenazioni circolari, i cui effetti sono chiaramente imprevedibili. La compilazione da parte dell'assembler può essere conclusa solo da un (EOF) o da un comando .END.

3.0 Generazione di macro
Le macro hanno la struttura generale mostrata nel seguente codice:
(etichetta) .MAC nome della macro
Testo della Macro
(etichetta) .MND
Il comando .MAC definisce una macro con il nome di macro definito, e crea fino a nove parametri per quella macro. L'utente non dichiara esplicitamente i parametri. Le macro possono contenere qualsiasi testo tranne i comandi .MAC e .MND. Il comando .MND identifica la fine della definizione di macro. Le etichette su .MAC e .MND sono opzionali, come indicato dalle parentesi. Esse si riferiscono rispettivamente alla prima istruzione o comando generata ed all'istruzione o comando immediatamente successiva all'ultima.
Per chiamare una macro l'utente deve semplicemente fornire il nome della macro e tutti i parametri necessari, come esemplificato nella linea seguente:
nomemacro param1,param2,param3...
Il nome della macro deve essere delimitato da uno spazio prima del primo parametro, come si vede dall'esempio sopra.
Nel testo della definizione di macro, un parametro è indicato dal simbolo temporaneo '?' seguito da un numero da 1 a 9; ?3 indica pertanto il parametro 3. Durante la compilazione, se l'assembler incontra la chiamata di una macro, il testo incluso in quella macro è inserito nel codice oggetto in quel punto ed i nomi dei parametri al punto di chiamata sono sostituiti ai simboli temporanei. Se l'utente non fornisce il nome di un parametro quando

chiama una macro, l'assembler genera un nome per quel parametro (se uno è necessario), per la durata della chiamata stessa.
Per fare un breve esempio, supponiamo di voler incrementare un numero in doppia precisione (16 bit). La definizione di macro da fare è questa:

```
.MAC DPINC ; incremento in doppia PR.
INC ?1
BNE ?2
INC ?1 + 1
?2 .MND
```

Quando la macro è chiamata per incrementare la variabile COUNT, bisogna immettere una linea come la seguente:

```
DPINC COUNT
```

Questo genera il seguente codice:

```
INC COUNT
BNE L001
INC COUNT+1
L001
```

In questo esempio, l'etichetta interna 'L001' è generata automaticamente durante la compilazione della macro. Chiamate successive producono etichette distinte, seguendo la progressione L002, L003, etc... Se il programmatore fornisce un secondo parametro nella linea di chiamata, invece di lasciare quel parametro vuoto, l'etichetta interna viene chiamata come quel parametro, invece che L001.
Una macro può chiamare un'altra macro, ma bisogna fare attenzione che l'annidamento non superi otto livelli.

Parametri vuoti
I parametri vuoti nelle linee di chiamata sono evidenziati da virgole:

```
FUNCTN AA,,,DD
;i parametri ?2 e ?3 sono vuoti, come sono vuoti quelli dal ?5 al ?9.
FUNCTN ,,CC,,EE
;sono vuoti ?1, ?2 e ?4, così come da ?6 a ?9.
```

Nomi concatenati
I parametri delle macro vanno scritti senza spazi né davanti né dietro, in modo che un parametro possa essere usato per creare nuovi nomi di variabili e per allocare spazio per le variabili.

```
.MAC DICH ; spazio necessario
XX?1 .WOR 0
* = * + ?2
.MND
DICH AA,5 ; chiamata alla macro
XXAA .WOR 0
* =* + 5 ; così viene riscritta
DICH A2, 10 ; seconda chiamata
XXA2 .WOR 0
* = * + 10 ; così viene riscritta la seconda chiamata
```

Nota che non ci sono spazi nelle etichette XXAA e XXA2.

Espressioni come parametri
I parametri della macro possono essere espressioni arbitrarie che non contengono virgole, punti e virgola o spazi. Quando le espressioni sono inserite nella definizione di macro, l'espressione deve avere un significato per l'assembler.

```
.MAC minore ; se l'acc.<?1 allora ?2
CMP ?1
BCS ?2
.MND
MINORE XX+5,EXIT
;confronta con la loc. XX+5
MINORE #$F3,EXIT
;confronta con $F3
MINORE (XX,1),EXIT
;illegale - c'è una virgola!
```

Formato di output dell'assembler
Fai attenzione: il macro assembler usa regole differenti dal precedente assembler Commodore per decidere come formattare una linea di stampa. La nuova regola è la seguente: un identificatore che inizia alla colonna 1 è stampato come un'etichetta, altrimenti è considerato un'istruzione.
Questa regola è identica a quella usata dall'editor del comando FORMAT. Il comando FORMAT permette all'utente di vedere il formato finale stampato del file inserito.

4.0 File di output generati dall'assembler
Ci sono tre tipi di file di output generati dall'assembler. Ogni file è opzionale e può essere creato tramite l'uso del comando .OPT.
Il file di listato contiene il listato del programma con gli errori e la tavola dei simboli. Il file di errore contiene tutte le linee errate e gli errori (come nel file di listato). Il file di interfaccia contiene il codice oggetto per il caricatore.
Il file di riferimento incrociato può essere generato opzionalmente dall'assembler. Questo file è usato dal programma di riferimento incrociato per stampare un prospetto che mostri tutte le variabili, i loro indirizzi dichiarati e tutti i numeri di linea in cui esse sono usate.

File di listato
Il file di listato è prodotto ove non sia usata

l'opzione NOLIST nel comando .OPT. Questo file è formato da due sezioni: listato del programma e degli errori; tavola dei simboli.
* Listato del programma e degli errori.
Contiene codice sorgente insieme con il codice compilato. Messaggi di errore e di pericolo appaiono dopo ogni comando errato. (Una spiegazione dei messaggi di errore è riportata nell'Appendice VI). Alla fine del programma è mostrato il conteggio degli errori e dei pericoli riscontrati.
* Tavola dei simboli.
La tavola dei simboli viene prodotta allorché non venga usata l'opzione NOSYM. Contiene la lista di tutte le variabili e le costanti usate nel programma, ed i loro rispettivi indirizzi.

File di interfaccia
Questo file non contiene il codice oggetto vero e proprio, ma dati che possono essere letti dal caricatore e trasformati in codice macchina. Il formato del primo record e di tutti i successivi, tranne l'ultimo, è questo:
; n1n0 a3a2a1a0 (d1d0) 1 (d1d0) 2.. (d1d0) 23 x3x2x1x0
Dove siano valide le seguenti regole:
1. Tutti i caratteri (n,a,d,x) e i caratteri ASCII da A a F, rappresentano una cifra esadecimale.
2. Il punto e virgola indica l'inizio di ogni record.
3. n1n0
numero di byte di dati in questo record (in esadecimale). Ogni coppia di caratteri esadecimali (d1d0) rappresenta un singolo byte.
4.a3a2a1a0
Indirizzo iniziale esadecimale del record. a3 rappresenta i bit d'indirizzo dal 15 al 12, a2 quelli dall'11 all'8, etc.. Il byte rappresentato da (d1d0) 1 è posto all'indirizzo a3a2a1a0, (d1d0) 2 è posto a a3a2a1a0 + 1, etc..
5 (d1d0)
Coppia di cifre esadecimali che rappresentano un byte (8 bit) di dati. (d1 rappresenta i 4 bit alti, d2 i 4 bit bassi). Sono permessi un massimo di 24 byte per ogni record.
6. x3x2x1x0
Check sum (somma di controllo) del record. Questa è la somma esadecimale di tutti i caratteri del record, inclusi n1n0 e a3a2a1a0, esclusi il punto e virgola iniziale ed il check sum stesso. Per generare il check sum ogni byte di dati (rappresentato da due caratteri ASCII), viene trattato come otto bit binari. La somma binaria di questi byte a 8 bit è troncata a 16 bit binari (4 cifre hex), ed è poi rappresentata nel record come quattro caratteri ASCII (x3x2x1x0).
Il formato dell'ultimo record in un file è il seguente:
; 00 c3c2c1c0 x3x2x1x0
1. ; 00
In questo record ci sono zero byte di dati; lo zero identifica l'ultimo record del file.
2. c3c2c1c0
Questo rappresenta il numero totale dei record nel file, escluso l'ultimo.
3. x3x2x1x0
Check sum di questo record.

5.0 Comandi Basic addizionali per la gestione del disco
Sul disco c'è un programma che ti aiuta nelle funzioni di gestione del disco (copiare, cancellare, cambiare nome, leggere la directory, inizializzare il disk drive, controllare lo stato del disco e naturalmente caricare e lanciare i programmi dal disco). I comandi che questo programma fornisce sono semplici, rapidi e molto utili.

5.1 Caricare il programma DOS WEDGE 64
Quando questo programma è caricato, si inserisce tra il sistema operativo ed il BASIC; esso controlla ogni carattere inserito dalla tastiera, prima di passarlo all'interprete BASIC, per individuare i comandi che lo interessano. (Questo è possibile inserendosi nella routine CHRGET di pagina zero). Per caricare il programma battete:
LOAD "DOS WEDGE64", 8,1
e premete RETURN. Questo carica un programma che a sua volta attiva il programma Wedge. Una volta caricato batti RUN e RETURN prima di togliere il dischetto dal drive. Non appena il programma viene lanciato, appare un messaggio di copyright.

5.2 Usare il programma DOS WEDGE64
Il programma supporta tutti i comandi inclusi nel BASIC (copia, cancella, cambia il nome, formatta), un comando per leggere la directory (senza modificare la memoria) e comandi per caricare e lanciare programmi. Esso ti viene anche in aiuto con la capacità di creare e gestire volumi di file (la creazione di volumi ti permette di raggruppare alcuni programmi), e la capacità di compiere operazioni usando nomi collettivi per i file (ad esempio, tutti i file il cui nome comincia con un certo carattere).
Ogni comando comincia con un carattere spe-

ciale, come specificato nella sezione 5.3. Il carattere usato dipende dal comando. I simboli ' @ ' e '>' sono usati indifferentemente per introdurre un comando di gestione del disco o di lettura della directory. Essi sono anche usati per resettare o reinizializzare il drive, e per uscire dal DOS WEDGE. Il simbolo ' ↑ ' (freccia in alto) è usato per introdurre il comando atto a caricare un programma (all'indirizzo BASIC 'Inizio del testo') e a lanciarlo automaticamente. Il simbolo / (barra obliqua discendente a sinistra) è usato per introdurre il comando che carica un programma all'indirizzo BASIC 'inizio del testo'. Il simbolo % (di percentuale) è usato per introdurre il comando che carica un programma al suo indirizzo di caricamento. Infine, il simbolo '←' (freccia all'indietro) è usato per introdurre il comando che salva i file su disco.

Comandi del programma DOS WEDGE64

Nelle pagine seguenti è fornita la descrizione di ogni comando. L'Appendice IX contiene un breve sommario di tutti i comandi del programma DOS WEDGE64.

@

Battendo questo comando da solo si viene informati sullo stato del disco. Questo svolge la stessa funzione delle seguenti linee BASIC:

```
10 OPEN 15,8,15
20 INPUT#15,A,B$,C,D
30 PRINT A;B$;C;D
```

@ $(drive):(nomefile) (*) ([volume])

Questo comando legge la directory del disk drive specificato e la stampa sullo schermo. Se viene specificato (nomefile), viene mostrato solamente il file avente quel nome. Se viene specificato (*), vengono mostrati tutti quei file il cui nome inizia con i caratteri specificati in (nomefile). Se viene specificato [volume], vengono mostrati solo i file contenuti in quel volume.

@ N(drive):nomedisco,id

Questo comando formatta un disco usando il nome e l'id specificati.

@ R(drive): nuovofile([volume]) = vecchiofile([volume])

Questo comando cambia il nome del file 'vecchiofile' in 'nuovofile'.

@ C(drive): nuovofile([volume]) = vecchiofile([volume])

Questo comando copia il file specificato, in 'vecchiofile' con il nome 'nuovofile'. Se viene specificato [volume], il nuovo file viene generato in quel volume.

@ S(drive):nomefile(*)([volume])

Questo comando cancella il file il cui nome è specificato in 'nomefile'. Se viene specificato (*), vengono cancellati tutti i file il cui nome inizia con i caratteri contenuti in 'nomefile'. Se viene specificato [volume], vengono cancellati solo i file contenuti in quel volume.

@UI(drive)

Questo comando resetta il DOS.

@ I(drive)

Questo comando inizializza il disk drive.

@Q

Questo comando esce dal programma Wedge.

/nomefile

Questo comando carica il file specificato da 'nomefile'. Ad esempio:

/ASM.C64

Fa sì che il programma chiamato 'ASM.C64' venga caricato in memoria. Questo comando è l'equivalente della linea BASIC:

LOAD "ASM.C64",8

Ricorda che questo comando è utilizzabile solo per caricare programmi in BASIC, o programmi in linguaggio macchina che vengano lanciati da BASIC, in quanto il computer ignora l'indirizzo di caricamento proprio del file, e lo carica sempre all'indirizzo BASIC 'Inizio del Testo'.

%nomefile

Questo comando carica il file specificato da 'nomefile' a partire dall'indirizzo di caricamento proprio del file stesso. È l'equivalente della seguente linea BASIC:

LOAD "nomefile",8,1

dove 'nomefile' è il nome del programma da caricare.

↑ nomefile

Questo comando permette all'utente di caricare e lanciare il programma specificato da 'nomefile', ed è equivalente a:

LOAD "nomefile", 8,1

seguito dal comando BASIC 'RUN'.
ATTENZIONE: anche questo comando permette solamente di caricare programmi in BASIC, o programmi in linguaggio macchina, lanciati da BASIC.

← nomefile

Questo comando salva il programma specificato da 'nomefile' su disco.

6.0 Creare ed editare un file sorgente

L'editor è usato per creare e modificare file sorgenti per l'assembler. Esso mantiene tutte le caratteristiche dell'editor a tutto scher-

mo del BASIC, ed inoltre permette la numerazione automatica delle righe (AUTO), la ricerca di stringhe (FIND), la sostituzione (CHANGE), la cancellazione di righe in un intervallo (DELETE) e la rinumerazione automatica (NUMBER). Altri comandi includono GET, PUT, BREAK, KILL e FORMAT. Tutti i comandi sono elencati in dettaglio nel sommario alla fine di questa sezione.
I comandi dell'editor operano in modo simile ai comandi già esistenti nell'editor del BASIC. Per fare pratica vi suggeriamo di creare alcuni file di esempio usando i comandi dell'editor. I file di dati sui quali opera l'assembler sono composti di caratteri ASCII CBM, con ogni linea terminante con un carriage return (ritorno di carrello). La sola restrizione nei file di dati è nel nome: a causa del metodo con cui i nomi vengono esaminati dall'assembler, infatti, non possono contenere alcuno spazio. I file sono sequenziali e devono terminare con un byte zero ($00). Quando richiedete la directory, questi file sono mostrati come file di tipo SEQ. Il formato di ogni file è sequenziale quando questo termina con un byte zero ($00).

6.1 Caricare il programma Editor 64
L'editor deve essere caricato con il comando BASIC LOAD:

LOAD "EDITOR64",8,1

oppure %0:EDITOR64 (se il programma wedge è abilitato).
Per lanciare l'editor battete 'SYS49152'. Dopo aver battuto il comando SYS, l'editor risponde con un messaggio che indica che l'editor 64 è stato attivato. A questo punto batti NEW per resettare il puntatore del testo. Ora sei pronto a creare o modificare un file sorgente.

6.2 Uso del programma Editor 64
Quando il programma Editor64 è attivo, qualsiasi comando BASIC battuto così:

10 FOR A = 1 TO 10

non viene tokenizzato (convertito in token, parola chiave BASIC che occupa un solo byte). Per questo non si possono battere linee BASIC se l'editor è attivato. Per superare questo problema è sufficiente disattivare l'editor con il comando 'KILL', o resettare il computer per tornare al BASIC.
I file sorgente vengono caricati con il comando 'GET'. Non appena il file è caricato in memoria, l'editor genera automaticamente i numeri di linea, partendo da 1000. Dopo aver editato un file, assicurati che l'ultima linea contenga un comando. FILE o .END. Poi salva il file su disco con il comando 'PUT'.
Importante: assicurati di salvare il file con 'PUT' prima di caricare l'assembler, altrimenti il file viene perduto.
Nell'Appendice VII è contenuto il sommario dei comandi del programma Editor64.
Comandi del programma Editor64.

AUTO
Il comando AUTO genera automaticamente i numeri di linea quando si inserisce un nuovo file sorgente. Per attivare il comando AUTO batti:
AUTO n1
dove n1 è l'incremento opzionale tra i numeri di linea stampati. Per disabilitare la funzione AUTO, batti AUTO senza alcun operando.

CHANGE
Il comando CHANGE individua una stringa e la sostituisce automaticamente con un'altra. Il formato di questo comando è il seguente:

CHANGE/str1/str2/,(n1-n2)

/ delimita str1 e str2 (usa qualsiasi carattere che non sia in nessuna delle stringhe)
str1 stringa da cercare
str2 stringa sostitutiva
,n1-n2 parametro di intervallo. Il formato è il medesimo del comando LIST in BASIC. Se viene omesso, il comando opera in tutto il file (Opzionale).

CPUT
Il comando CPUT salva su disco il file sorgente per la successiva compilazione, eliminando gli spazi non necessari. La sintassi è la medesima del comando PUT.

DELETE
La funzione DELETE permette all'utente di cancellare più linee contemporaneamente. Inserisci l'intervallo di linee da cancellare (da n1 a n2). Il formato è lo stesso del comando LIST in BASIC.

DELETE n1-n2

Per cancellare una singola linea batti il numero di linea su di una linea vuota e premi RETURN.

FIND
Il comando FIND è usato per cercare ed individuare una specifica stringa di caratteri. Ogni ricorrenza della stringa viene stampata sullo schermo. Si può fermare la stampa con la bar-



ABBREVIAZIONI

ABS	A{shft B}	182
AND	A{shft N}	175
ASC	A{shft S}	198
ATN	A{shft T}	193
CHR$	C{shft H}	199
CLOSE	CL{shft O}	160
CLR	C{shft L}	156
CMD	C{shft M}	157
CONT	C{shft O}	154
COS	none	190
DATA	D{shft A}	131
DEF	D{shft E}	150
DIM	D{shft I}	134
END	E{shft N}	128
EXP	E{shft X}	189
FN	none	165
FOR	F{shft O}	129
FRE	F{shft R}	184
GET	G{shft E}	161
GET#	none	161
GO	none	203
GOSUB	GO{shft S}	141
GOTO	G{shft O}	137
IF	none	139
INPUT	none	133
INPUT#	I{shft N}	132

INT	none	181
LEFT$	LE{shft F}	200
LEN	none	195
LET	L{shft E}	136
LIST	L{shft I}	155
LOAD	L{shft O}	147
LOG	none	188
MID$	M{shft I}	202
NEW	none	162
NEXT	N{shft E}	130
NOT	N{shft O}	168
ON	none	145
OPEN	O{shft P}	159
OR	none	176
PEEK	P{shft E}	194
POKE	P{shft O}	151
POS	none	185
PRINT	?	153
PRINT#	P{shft R}	152
READ	R{shft E}	135
REM	none	143
RESTORE	RE{shft S}	140
RETURN	RE{shft T}	142
RIGHT$	R{shft I}	201
RND	R{shft N}	187
RUN	R{shft U}	138
SAVE	S{shft A}	148
SGN	S{shft G}	180
SIN	S{shft I}	191
SPC(	S{shft P}	166
SQR	S{shft Q}	186
STATUS	ST	
STEP	ST{shft E}	169
STOP	S{shft T}	144
STR$	ST{shft R}	196
SYS	S{shft Y}	158
TAB(	T{shft A}	163
TAN	none	192
THEN	T{shft H}	167
TIME	TI	
TIME$	TI$	
TO	none	164
USR	U{shft S}	183
VAL	V{shft A}	197
VERIFY	V{shft E}	149
WAIT	W{shft A}	146

DISK COMMANDO

Format a New Diskette:

```
OPEN 15,8,15
PRINT#15,"N0:name,id"
CLOSE 15
```

Initialize:

```
OPEN 15,8,15
PRINT#15,"I"
CLOSE15
```

Validate:

```
OPEN 15,8,15
PRINT#15,"V"
CLOSE15
```

Save a Program:

```
SAVE"name",8
```

Save with Replace:

```
SAVE"@0:name",8
```

Load Basic Program:

```
LOAD"name",8
```

Load Machine Lang. Program:

```
LOAD"name",8,1
```

List the Directory:

```
LOAD"$",8
LIST
```

Scratch a Program:

```
OPEN15,8,15
PRINT#15,"S0:name"
CLOSE15
```

Rename a Program:

```
OPEN 15,8,15
PRINT#15,"R0:
newname = oldname"
CLOSE15
```

Copy a Program:

```
OPEN 15,8,15
PRINT #15,"C0:
newname = 0:oldname"
CLOSE15
```

Read Error Channel:

```
10 OPEN15,8,15:
   INPUT#15,A,B$,
   C,D:PRINTA,B$,
   C,D:CLOSE15
RUN
```

SUONO

SI = 54265 [SID chip start]

V = voice number 1–3

Volume: POKE 54296,X 0–15

Notes:

Hi	POKE SI + V*7 + 1,X	1–255
Low	POKE SI + V*7,X	1–255
Wave:	POKE SI + V*7 + 4,X	1–255

X = 17 triangle X = 65 pulse

X = 33 sawtooth X = 129 noise

Amplitude: (for pulse wave)

Hi	POKE SI + V*7 + 3,X	0–255
Low	POKE SI + V*7 + 2,X	0–15

Attack/Decay:

POKE SI + V*7 + 5,(A*16 + D) A = 0–15 D = 0–15

Sustain/Release:

POKE SI + V*7 + 6,(S*16 + R) S = 0–15 R = 0–15

COLORE

COLOR	ASCII	POKE
BLACK	144	0
WHITE	5	1
RED	28	2
CYAN	159	3
PURPLE	156	4
GREEN	30	5
BLUE	31	6
YELLOW	158	7
ORANGE	129	8
BROWN	149	9
Lt. RED	150	10
GRAY	151	11
Med. GRAY	152	12
Lt. GREEN	153	13
Lt. BLUE	154	14
Lt. GRAY	155	15

ra spaziatrice. La stampa può poi essere ripristinata premendo ancora la barra spaziatrice, oppure fermata definitivamente con il tasto RUN/STOP.
Il formato del comando FIND è:

FIND/str1/str2/,(n1-n2)

/ delimitatore (usa un carattere che non sia in nessuna delle due stringhe)
str1 stringa da cercare
,n1-n2 parametro di intervallo. Uguale a quello del comando LIST in BASIC (Opzionale).

Stampa FORMATtata
Il comando FORMAT è usato per stampare testo in formato tabulato come l'assembler. Perché questa funzione lavori correttamente, devi battere gli mnemonici nella colonna due, o ad uno spazio di distanza dalle etichette.

FORMAT (n1-n2)

n1-n2 parametro di intervallo dello stesso formato di LIST (Opzionale)
Nota: questo comando ha gli stessi controlli di FIND. Ad esempio premi la barra spaziatrice per fermare e far ripartire la stampa ed il tasto RUN/STOP per abortirla.

GET
Questo comando serve a caricare da disco file sorgente di testo dell'assembler nell'editor. Esso può anche essere usato per aggiungere file ad altri file già in memoria.

GET "nomefile", (n1), (n2), (n3)

n1 comincia a caricare il file sorgente a partire da questa riga del file in memoria.
n2 numero di device (periferica). Il valore per difetto è 8 (Opzionale).
n3 indirizzo secondario. Il valore per difetto è 8 (Opzionale).
Nota: GET comincia a numerare le linee da 1000, con incrementi di 10. Se n1 è più grande di qualsiasi numero di linea in memoria, il file da caricare è concatenato alla fine del file corrente.

KILL
Questo comando disabilita l'editor. Per rilanciare l'editor, batti lo stesso comando usato per lanciarlo (SYS49152).

LIST
Il comando dell'editor LIST lavora come il medesimo comando BASIC.

List (n1)-(n2)

dove n1-n2 rappresenta l'intervallo di linee in cui il comando agisce. Sono considerati validi anche i parametri 'n1-' (che lista tutte le linee a partire dalla linea n1), e '-n2' (che lista le linee comprese tra la prima e n2).

NUMBER
La funzione NUMBER permette all'utente di rinumerare tutto o parte del file in memoria.

NUMBER (n1), (n2), (n3)

n1 numero della vecchia linea iniziale
n2 numero della nuova linea iniziale
n3 incremento di rinumerazione
Nota: tutti i parametri sono opzionali.

PUT
Il comando PUT salva il file sorgente su disco per la successiva compilazione. PUT può salvare tutto o parte del file residente in memoria.

PUT "nomefile", (n1-n2),(n3),(n4)

n1 numero della linea iniziale
n2 numero della linea finale
n3 numero di device, per difetto è 8
n4 indirizzo secondario, per difetto è 8
Tutti i parametri sono opzionali. Se vengono omessi tutti i parametri, l'intero file viene salvato su disco.

7.0 Compilare un file sorgente
Quando un file sorgente è pronto per essere compilato devi innanzitutto salvarlo su disco (usando il comando PUT). Assicurati di compiere questa operazione prima di caricare il programma assembler, altrimenti son dolori. Una volta fatto questo, potrai caricare l'assembler, che risiede nella stessa zona dei programmi BASIC.

7.1 Caricare il programma Assembler 64
Per caricare l'assembler batti:

LOAD "ASSEMBLER64",8

(o /ASSEMBLER64 se è stato caricato il programma DOS WEDGE64)
Quando il caricamento è completato, batti RUN e premi RETURN.
L'assembler stampa una prima richiesta.

7.2 Usare il programma Assembler64
Quando si sta compilando un programma, si possono creare due tipi di file: il primo è un file oggetto che contiene i dati necessari per creare un programma in codice macchina mediante il loader (caricatore). Il nome di questo file è specificato dall'utente prima che cominci la compilazione. Gli altri file sono file di cross reference. Il nome di questi file è ge-

nerato automaticamente dall'assembler, ed è nel formato 'XXLLOOOO' e 'XXFFOOOO'.
Bisogna notare che l'assembler non riscrive mai nessuno di questi file (cancellando i file aventi il medesimo nome), quindi se vuoi usare lo stesso nome per i file oggetto ogni volta che assembli un programma devi cancellare dal disco i vecchi file oggetto prima di caricare l'assembler. Lo stesso discorso vale naturalmente anche se vuoi creare nuovi file di cross reference.
Sebbene ti sia data l'opzione di creare sia un file oggetto, sia un file di cross reference, solo una di queste opzioni può essere scelta (a causa del numero di file aperti contemporaneamente). Se vuoi entrambi i file, lancia l'assembler una volta con l'opzione file oggetto, un'altra volta con l'opzione cross reference.
Quando l'assembler parte, la prima richiesta è:

NOME OGGETTO (RET/NO o D: NOME):

Se vuoi che l'assembler crei un file oggetto, inserisci il nome del file e premi RETURN, altrimenti premi solo RETURN. Poi viene stampata la richiesta:

STAMPANTE (RET/S o N)?

Se vuoi una copia su stampante, premi Y e RETURN o semplicemente RETURN, altrimenti, premi N seguito da RETURN. Questo fa sì che il listato sia stampato sullo schermo.
Poi viene stampata la richiesta:

CROSS REFERENCE (RET/o Y)?

Se vuoi che sia creato un file di cross reference, premi Y seguito da RETURN, altrimenti solo RETURN. Infine viene stampata la richiesta:

NOME PRG SORGENTE?

Batti il nome del file sorgente che vuoi assemblare.
Ricevuta quest'ultima informazione, l'assembler comincia a lavorare. Se durante la compilazione la tabella dei simboli trabocca, il processo di compilazione si ferma.

Fermare l'assembler

Quando l'assembler sta lavorando, le operazioni possono essere fermate premendo il tasto RUN/STOP. Se questo viene fatto, il processo di compilazione viene fermato ed il programma attende che l'utente decida se continuare o porre fine alla compilazione. Premi il tasto B per porre fine alla compilazione e per tornare al BASIC, un qualsiasi altro tasto per continuare. Questa possibilità è utile soprattutto per chi non possiede la stampante, perché permette di esaminare il listato sullo schermo durante la compilazione.

File di riferimento incrociato (Cross reference)

Se scegli di produrre un riferimento incrociato verranno creati due file, come accennato sopra. Per esaminare o produrre un listato su stampante del riferimento incrociato, devi prima caricare un apposito programma, in questo modo:

LOAD "CROSSREF64",8

Quando il programma è stato caricato, batti RUN seguito da RETURN ed il programma stampa la seguente richiesta:

NOME OGGETTO (RET o D: NOME):

Premi RETURN se vuoi un listato su stampante, altrimenti batti N seguito da RETURN.

8.0 Caricare un file oggetto

L'assembler del Commodore 64 produce file trasportabili in formato ASCII, che non possono essere eseguiti direttamente. È necessario infatti che questi siano caricati mediante un particolare programma chiamato LOADER (caricatore) per poter essere lanciati.

8.1 Caricare il programma LOADER

Nel disco sono incluse due versioni del LOADER, posizionate in differenti aree della memoria RAM. Questo permette all'utente di caricare programmi ovunque nella RAM usando il caricatore corretto. Per caricare uno dei due LOADER, batti:

LOAD "nomefile",8,1

dove "nomefile" è il nome del programma da caricare. La seguente tabella mostra i nomi, gli indirizzi di caricamento ed i comandi per lanciare ognuno dei due caricatori.

Nome	Indirizzo	Comando di lancio
LOLOADER64	$0800	RUN
HILOADER64	$C800	SYS 51200

8.2 Usare i programmi di caricamento (Loader)

Entrambi i programmi di caricamento (HILOADER e LOLOADER) sono lunghi circa 512 byte e operano allo stesso modo. Quando viene attivato, il caricatore stampa un messaggio di copyright e richiede all'utente di inserire uno scostamento di caricamento. Lo scostamento è usato per porre il codice oggetto ad un indirizzo differente da quello in cui è stato assemblato. Questo permette all'utente di assemblare per una zona in cui non c'è RAM e di caricare in una zona RAM.

L'oggetto può in questo modo essere programmato in un'EPROM etc. Lo scostamento è un indirizzo esadecimale di due byte che viene sommato agli indirizzi del programma. Se l'indirizzo del programma, sommato allo scostamento, dà un valore maggiore di $FFFF, l'indirizzo riparte da $0000. Gli esempi seguenti mostrano come lavora lo scostamento:

Ind. di prog.	Scost.	Ind. del codice oggetto
$0400	$0000	$0400
$3000	$0000	$3000
$0400	$2000	$2400
$9000	$9000	$2000
$E000	$4000	$2000

Dopo che è stato inserito lo scostamento, il caricatore richiede all'utente il nome del file oggetto da caricare. Il caricatore poi inizializza il drive, cerca il file e comincia il caricamento. Quando il caricamento è completato, il caricatore stampa il messaggio.
FINE CARIC. e ritorna al BASIC. Possono verificarsi tre errori durante il caricamento:
ERRORE CONTO REC
(conteggio dei record errato)
CAR. NON -RAM
(caricamento in una zona di memoria a sola scrittura)
ERRORE SOMMA
(errore nella somma di controllo)
Gli errori sono considerati fatali; il caricamento termina, il file oggetto viene chiuso e viene restituito il controllo al BASIC.

9.0 Controllare e correggere con i programmi monitor

Ci sono due monitor di linguaggio macchina sul disco: Monitor$8000 e Monitor$C000. La sola differenza tra di loro è l'area di memoria in cui risiedono. Il programma Monitor$8000 risiede alla locazione di memoria $8000 ed il programma Monitor$C000 risiede alla locazione $C000. Sono incluse le due versioni nel caso una di loro interferisca con le locazioni in cui volete che risieda il programma da controllare. Per caricare ed attivare il monitor appropriato battete:

LOAD "MONITOR$8000",8,1
(monitor a $8000)
SYS 32768

oppure

LOAD "MONITOR$C000",8,1
(monitor a $C000)
SYS 49152

9.2 Usare i programmi Monitor

I programmi Monitor, non appena caricati, mostrano i registri della CPU, stampano un punto e fanno lampeggiare il cursore. Il punto serve ad informarti che il programma è in attesa di un tuo comando. I comandi sono descritti uno per uno nelle pagine che seguono. Nell'appendice VIII c'è inoltre un sommario dei comandi dei programmi Monitor.
Per uscire dal programma Monitor devi resettare il computer.

9.3 Comandi del programma Monitor

Comando A

Fine: inserire una linea di codice assembler
Sintassi: A (indirizzo) (codifica) (operando)
(indirizzo): un numero esadecimale di quattro cifre indicante la locazione di memoria in cui porre la codifica.
(codifica mnemonica): una codifica mnemonica standard nel linguaggio assembler MOS, ad Es. LDA, STX, ROR, etc., come definito nell'Appendice IV.
(operando): l'operando, quando è richiesto, può essere in uno qualsiasi dei modi di indirizzamento legali. Per i modi di pagina zero è necessario un numero esadecimale di quattro cifre il cui valore sia minore o uguale a $FF. Per gli indirizzi in pagine diverse dalla zero è necessario un numero esadecimale di quattro cifre il cui valore sia minore o uguale a $FFFF. Un RETURN viene usato per indicare la fine della linea in linguaggio assembler. Se vengono riscontrati errori, un punto di domanda indica l'errore, e viene stampato un punto sulla linea successiva. L'editor di testo può essere usato per correggere gli errori sulla linea originale.

Dopo che una linea di codice è stata assemblata con successo, l'assembler stampa la successiva locazione legale per un'istruzione, in modo che non ci sia bisogno di battere 'A' e l'indirizzo più di una volta, quando si inseriscono programmi in linguaggio assembler nel Commodore 64. Per uscire da questo modo, premi RETURN dopo la scritta 'A'. generata dal computer.

```
Esempio: .A 1200 LDX #$00
         .A 1202
```

Comando C (compare)

Fine: confrontare due zone di memoria.
Sintassi: C (ind. iniziale) (ind. finale) (ind. comparando)
(indirizzo iniziale): numero esadecimale di quattro cifre che indica l'indirizzo iniziale della prima delle due aree di memoria.

(indirizzo finale): numero esadecimale di quattro cifre che indica l'indirizzo finale della prima delle due aree di memoria.
(indirizzo comparando): numero esadecimale di quattro cifre che indica l'indirizzo iniziale della seconda area di memoria.
I campi degli indirizzi devono essere separati da un delimitatore valido, come uno spazio o una virgola. Se le due zone di memoria sono uguali, il programma stampa un punto. L'indirizzo di ogni byte differente nelle due zone viene stampato sullo schermo.

Comando D (disassemble)

Fine: disassemblare del codice macchina in linguaggio assembler (è il processo inverso della compilazione)

Sintassi: D (indirizzo1) (indirizzo 2)

(indirizzo 1): l'indirizzo inziale del codice da disassemblare espresso da un numero esadecimale di quattro cifre.

(indirizzo 2): indirizzo finale (opzionale) del codice da disassemblare, espresso da un numero esadecimale di quattro cifre.

I campi di indirizzo devono essere separati da un delimitatore valido, come uno spazio o una virgola. Il formato del disassemblato è solo leggermente differente da quello dell'assembler che si inserisce da tastiera. La differenza sta nel fatto che il primo carattere di un disassemblato è una virgola invece di una 'A', per una migliore leggibilità.

Un listato disassemblato può essere modificato usando l'editor di schermo. Fai qualsiasi modifica allo mnemonico e/o all'operando e premi RETURN. Questo inserisce la linea e chiama l'assembler per ulteriori modifiche.

Un disassemblato può essere fatto scorrere su e giù per lo schermo mediante i controlli del cursore. Quando una linea di disassemblato è in fondo allo schermo, la pressione del tasto 'cursore in basso' provoca lo scorrimento di una linea verso l'alto, per permettere la visualizzazione di un'altra linea di codice. Questo procedimento funziona anche per procedere a ritroso in un disassemblato, andando in cima allo schermo e premendo il tasto 'cursore in alto'.

```
Es. D 1000 1400
; 1000 LDA # $00
; 1002 ???
; 1003 BNE $F1030
```

Comando F (fill)

Fine: riempire un insieme di locazioni con un determinato byte.

Sintassi: F (indirizzo1) (indirizzo2) (byte)

(indirizzo1): prima locazione da riempire con il valore specificato da (byte)

(indirizzo2): ultima locazione da riempire con il valore specificato da (byte)

(byte): numero esadecimale di due cifre da scrivere nelle locazioni di memoria consecutive specificate.

Questo comando è utile per inizializzare strutture di dati o qualsiasi altra area di RAM.

Es. F 0400 0518 EA

Riempe le locazioni di memoria da $0400 a $0518 con il valore $EA (l'istruzione NOP)

Comando G (go)

Fine: cominciare l'esecuzione di un programma all'indirizzo specificato.

Sintassi: G (indirizzo)

(indirizzo): argomento opzionale che specifica il nuovo valore del contatore di programma e l'indirizzo dal quale deve partire l'esecuzione. Quando l'indirizzo viene omesso l'esecuzione parte dal valore corrente del contatore di programma (che può essere visionato con il comando R)
Il comando G ripristina tutti i registri e comincia l'esecuzione all'indirizzo iniziale specificato. Si raccomanda la prudenza nell'usare questo comando. (Sarebbe saggio porre un punto di uscita [breakpoint] da qualche parte nella linea dell'esecuzione del programma, per prevenire la perdita del controllo del sistema operativo.)

Es. G 040C

L'esecuzione comincia alla locazione $040C.

Comando H (hunt)

Fine: cerca nella memoria tutte le ricorrenze di un insieme di byte in un intervallo specificato.

Sintassi: H (indirizzo1) (indirizzo2) (dati)

(indirizzo1): indirizzo iniziale della ricerca

(indirizzo2): indirizzo finale della ricerca.

(dati): insieme di dati da cercare (possono essere valori esadecimali o una stringa ASCII)

Una stringa ASCII è individuata da una virgoletta semplice che precede il primo carattere (ad es. 'STRINGA). Gli argomenti esadecimali di due cifre devono essere separati l'uno dall'altro da uno spazio.

Es. H C000 FFFF 'CIAO ; cerca la stringa ASCII CIAO
H A000 A101 A9 FF 4C ; cerca i valori A9, FF, 4C, in questa sequenza

Comando I (interrogate)

Fine: mostra la memoria in formato ASCII nell'intervallo di locazioni specificato.

Sintassi: I (indirizzo1) (indirizzo2)

(indirizzo1): indirizzo della prima locazione.

(indirizzo2): indirizzo dell'ultima locazione.

I caratteri ASCII sono mostrati in video inverso per contrasto con i dati esadecimali visualizzati sullo schermo.

Lo schermo può venire fatto scorrere usando il tasto up/down del cursore. Questo permette di continuare a cercare tra i parametri di ricerca.

Nota: quando un carattere non è stampabile, al suo posto viene visualizzato un punto (.).

Es. I C000 C020

Visualizza in inverso i caratteri compresi tra $C000 e $C020

Comando L (load)

Fine: caricare un file da cassetta o da disco.

Sintassi: L "nomefile", (device)

nomefile: qualsiasi nome di file legale sul Commodore 64

(device): un byte di due cifre indicante la periferica da cui caricare il file.

01 è da cassetta
08 è da disco (o 09, etc)

Il comando Load fa sì che un file venga caricato in memoria. L'indirizzo di partenza è contenuto nei primi due byte del file stesso (in un file PGM). In altre parole un comando LOAD carica sempre i file nella stessa zona da cui sono stati salvati. Questo è molto importante lavorando in linguaggio macchina, in quanto solamente pochi programmi sono completamente rilocabili. Il file viene caricato in memoria fino a che viene trovato il segnale di fine del file (EOF).

Es. L "SCREEN", 01 ; carica un file da cassetta
L "SCREEN", 08 ; carica un file da disco

Comando M (memory display)

Fine: visualizzare il contenuto della memoria nell'intervallo di locazioni specificato.

Sintassi: M (indirizzo1) (indirizzo2)

(indirizzo1): indirizzo di partenza

(indirizzo2): indirizzo finale (Opzionale. Se viene omesso vengono mostrati otto byte.)

La memoria è visualizzata nel seguente formato:

.:A048 7F E7 00 AA AA AE 02 FF

Il contenuto della memoria può essere modificato usando l'editor di schermo. Per editare, muovi il cursore ai dati da modificare, batti le correzioni necessarie e premi RETURN. Se c'è una locazione RAM errata o se si tenta di modificare una locazione ROM, viene mostrato un segnale di errore (?).

Come nei comandi D(DISASSEMBLE) e I(INTERROGATE), il contenuto dello schermo può venire fatto scorrere su e giú usando i controlli del cursore.

Es. M0000
.: 0000 4C 7F EF AA 00 02 F7 FF

Vengono visualizzati i primi otto byte di memoria.

Comando N (new locator)

Fine: rilocare riferimenti assoluti alla memoria sommando uno scostamento all'operando del codice.

Sintassi: N (ind1) (ind2) (scostamento) (ref1) (ref2) W

(ind1): indirizzo iniziale del codice da modificare

(ind2): indirizzo finale del codice da modificare

(scostamento): valore da sommare all'operando delle istruzioni. Il codice spostato da una zona più alta ad una più bassa della memoria ha bisogno di un valore particolare. Per ottenere questo valore bisogna sottrarre lo scostamento voluto da 65536 ($10000). Se ad es. vuoi spostare una parte di codice da $A000 (40960) a $0400 (1024), il valore dello scostamento è $6400 (25600), infatti 65536-(40960-1024) = 25600. Per capire meglio il signficato di questa operazione, devi ragionare con i numeri esadecimali: $A000 + $6400 è uguale a $10400, ma poichè il computer non possiede il bit nove, il risultato è $0400.

(ref1): ogni operazione lunga tre byte il cui operando sia maggiore o uguale a (ref1) e minore di (ref2) viene rilocata con uno scostamento uguale a (scostamento).
L'operando dell'istruzione lunga tre byte viene cioè rimpiazzato da 'operando + (scostamento)'.

(ref2): limite superiore degli operandi da rilocare (vedi ref1). Ogni operando con un valore maggiore o uguale a (ref2) non viene rilocato.

W: riloca le tabelle di WORD (opzionale). Se viene incluso W, tutte le coppie di byte vengono rilocate. La rilocazione diviene così indipendente dai dati.

Spesso è utile muovere una parte di codice da una zona di memoria ad un'altra (vedi il comando 'T') per fare spazio ad altro codice. Usando il comando 'N', il codice può essere modificato per funzionare al nuovo indirizzo.

Comando R (register display)

Fine: mostra i registri importanti del 6502: il registro di stato del programma, il contatore di programma, l'accumulatore, i registri indice X e Y, il puntatore dello stack.

Sintassi: R

Nota che il puntatore dello stack è visualizzato senza l'ottavo bit implicito. Dal momento che è stato menzionato l'ottavo bit del puntatore dello stack, è bene far notare un difetto del 6502. Quando viene eseguita un'istruzione PHP, sull'ottavo bit del puntatore di stack viene compiuta un'operazione di OR logico con il byte di stato, ed il risulatato viene immagazzinato nello stack con il bit quattro (il flag di break!) sempre settato. Per il 99,9% delle applicazioni, questo non crea problemi. Purtroppo però quando questo difetto viene a galla, causa problemi molto difficili da individuare.

Es. R
PC SR AC XR YR SP
.; 057F 01 02 03 04 FE

Comando S (save)

Fine: salvare il contenuto della memoria su registratore o su disco.

Sintassi: S "nomefile", (device), (indirizzo1), (indirizzo2) nomefile: qualsiasi nome legale per salvare dati.
Il nome del file deve essere racchiuso tra doppie virgolette; le virgolette semplici sono illegali.

(device): le due periferiche possibili sono il registratore a cassette ed il disk drive: per salvare su cassetta usa 01, per salvare su disco usa 08.

(indirizzo1): indirizzo iniziale della memoria da salvare.

(indirizzo2): indirizzo finale della memoria da salvare, più uno. Tutti i dati fino a quell'indirizzo, escluso il byte a quell'indirizzo, vengono salvati.

Il file creato da questo comando è un file di caricamento, ovvero contiene nei primi due byte l'indirizzo iniziale dei dati (indirizzo1). Il file può essere richiamato con il comando 'L'.

Es. S "GAME", 08,0400,0C00
Salva la memoria da $0400 a $0C00 su disco.

Comando T (transfer)

Fine: trasferire segmenti di memoria da una zona ad un'altra.

Sintassi: T (indirizzo1) (indirizzo2) (indirizzo3)

(indirizzo1): indirizzo iniziale dei dati da spostare

(indirizzo2): indirizzo finale dei dati da spostare

(indirizzo3): indirizzo inziale della nuova zona di memoria in cui vengono posti i dati

I dati possono essere spostati da una zona bassa ad una alta e viceversa. Segmenti di memoria di qualsiasi lunghezza possono essere mossi in avanti o all'indietro di un numero qualsiasi di byte.

Es. T 1400 1600 1401

Sposta i dati da $1400 fino a $1600 (incluso) un byte più in alto nella memoria.

APPENDICE III
DESCRIZIONE DEI FILE CONTENUTI SUL DISCO

ASSEMBLER64 - Vedi la sezione 7. Questo è l'assembler vero e proprio, che viene caricato nella zona bassa della memoria ed assembla i file creati con il programma EDITOR. Per caricare l'assembler batti LOAD "ASSEMBLER64", 8, oppure usa il comando di caricamento dal programma WEDGE; poi batti RUN. Qualsiasi file sorgente che non sia stato precedentemente salvato viene perduto, giacchè l'assembler viene allocato nella stessa zona di memoria da esso usata.

BOOT ALL - Questo programma carica e lancia il DOS WEDGE, il caricatore HILOADER e l'EDITOR allo stesso tempo. Questi tre programmi risiedono in aree di memoria differenti, in modo da permettere il loro utilizzo contemporaneo.

CROSSREF64 - Questo programma serve a stampare il listato di riferimento incrociato creato dall'assembler con un'apposita opzione al momento della compilazione. Il programma viene caricato nella parte bassa della memoria con il comando LOAD "CROSSREF64", 8 e viene lanciato con il comando RUN.

DOS 5.1. Questo file contiene il codice macchina per il programma WEDGE. Esso viene caricato automaticamente lanciando il programma WEDGE stesso.

DOS WEDGE64 - Vedi la sezione 5. Questo programma carica il DOS 5.1. È il primo programma su disco, così che può essere caricato con il comando LOAD "*", 8. Dopo che il programma è stato caricato, batti RUN.

EDITOR64 - Vedi la sezione 6. Questo programma è usato per creare e modificare i file di codice sorgente che devono essere assemblati. Per caricare l'editor, batti LOAD "EDITOR64",8,1. Dopo che il programma è stato caricato, batti SYS 49152 per lanciarlo. Poi batti NEW per azzerare i puntatori prima di creare o editare i file. Assicurati di salvare il codice sorgente usando il comando PUT prima di caricare l'assembler.

LOLOADER64 e HILOADER64 - Vedi la sezione 8. Questi due programmi sono usati per caricare i record sequenziali creati dall'assembler come codice oggetto. Quando uno di questi due programmi viene lanciato, esso carica il file oggetto nelle locazioni di memoria specificate, e lo trasforma in codice macchina pronto per essere eseguito. La sola differenza tra i due programmi consiste nel fatto che LOLOADER risiede all'indirizzo esadecimale $0800, mentre HILOADER risiede a $C800. Essi sono entrambi forniti nel caso che uno di loro interferisse con la locazione in cui volete porre il codice oggetto. Per caricare la versione posta a $0800, batti LOAD "LOLOADER64", 8 seguito da RUN; per caricare l'altra, batti LOAD "HILOADER64", 8 seguito da SYS 51200.

MONITOR $8000 e MONITOR $C000 - Vedi la sezione 9. Questi due monitor sono pressochè identici, e vengono usati principalmente per caricare e salvare file binari di codice macchina in forma eseguibile. Essi permettono anche

all'utente di esaminare e modificare i programmi senza dovere ripetere tutto il processo di compilazione. Il primo viene caricato all'indirizzo esadecimale $8000 battendo LOAD "MONITOR$8000",8,1 e viene lanciato con SYS 32768. Il secondo programma viene caricato all'indirizzo esadecimale $C000 battendo LOAD "MONITOR$C000",8,1 seguito da SYS 49152.

APPENDICE V
CODIFICHE DEI MICROPROCESSORI DELLA SERIE 6500

ADC Add with Carry to Accumulator -- somma con riporto all'accumulatore
AND "AND" to accumulator -- operazione logica AND
ASL Shift Left One Bit (Memory or Accumulator) -- sposta di un bit a sinistra (la memoria o l'accumulatore)
BCC Branch on Carry Clear -- salta se il flag di riporto è uguale a zero
BCS Branch on Carry Set -- salta se il flag di riporto è uguale a uno
BEQ Branch on Zero Result -- salta se il risultato è zero
BIT Test Bit in Memory with Accumulator -- controlla un bit di memoria con l'accumulatore
BMI Branch on Result Minus -- salta se il risultato è negativo
BNE Branch on Result not Zero -- salta se il risultato è diverso da zero
BPL Branch on Result Plus -- salta se il risultato è positivo
BRK Force an Interrupt or Break -- forza un interrupt
BVC Branch on Overflow Clear -- salta se il flag di overflow è uguale a zero
BVS Branch on Overflow Set -- salta se il flag di overflow è uguale a uno
CLC Clear Carry Flag -- poni il flag di riporto uguale a zero
CLD Clear Decimal Mode -- esci dal modo decimale
CLI Clear Interrupt Disable Bit -- poni il bit che disabilita gli interrupt uguale a zero
CLV Clear Overflow Flag -- poni il flag di overflow uguale a zero
CMP Compare Memory and Accumulator -- confronta la memoria con l'accumulatore
CPX Compare Memory and Index X -- confronta la memoria con l'indice X
CPY Compare Memory and Index Y -- confronta la memoria con l'indice Y
DEC Decrement Memory by One -- decrementa di uno la memoria
DEX Decrement Index X by One -- decrementa di uno l'indice X
DEY Decrement Index Y by One -- decrementa di uno l'indice Y
EOR Exclusive-OR Memory with Accumulator -- operazione logica di OR esclusivo
INC Increment Memory by one -- incrementa di uno la memoria
INX Increment Index X by one -- incrementa di uno l'indice X
INY Increment Index Y by one -- incrementa di uno l'indice Y
JMP Jump to New Location -- salto ad una nuova locazione
JSR Jump To New Location Saving Return Address -- salta ad una nuova locazione salvando l'indirizzo di ritorno
LDA Transfer Memory to Accumulator -- copia la memoria all'accumulatore
LDX Transfer Memory to Index X -- copia la memoria all'indice X
LDY Transfer Memory to Index Y -- copia la memoria all'indice Y
LSR Shift One Bit Right (Memory or Accumulator) -- sposta di un bit a destra (la memoria o l'accumulatore)
NOP No Operation -- Non fa niente
ORA "OR" Memory with Accumulator -- operazione logica OR
PHA Push Accumulator on Stack -- metti l'accumulatore nello stack
PHP Push Processor Status on Stack -- metti lo stato del processore nello stack
PLA Pull Accumulator from Stack -- prendi l'accumulatore dallo stack
PLP Pull Processor Status from Stack -- prendi lo stato del processore dallo stack
ROL Rotate One Bit Left (Memory Or Accumulator) -- ruota di un bit a sinistra (la memoria o l'accumulatore)
ROR Rotate One Bit Right (Memory or Accumulator) -- ruota di un bit a destra (la memoria o l'accumulatore)
RTI Return from Interrupt -- ritorna da un interrupt
RTS Return from Subruoutine -- ritorna da una subroutine
SBC Subtract Memory and Carry from Accumulator -- sottrai la memoria ed il riporto dall'accumulatore
SEC Set Carry Flag -- Poni il flag di riporto uguale a uno
SED Set Decimal Mode -- inserisci il modo decimale
SEI Set Interrupt Disable State -- disabilita gli interrupt

STA Store Accumulator in Memory – immagazzina l'accumulatore in memoria
STX Store Index X in Memory -- immagazzina il registro X in memoria
STY Store Index Y in Memory -- immagazzina il registro Y in memoria
TAX Transfer Accumulator to Index X -- trasferisci l'accumulatore all'indice X
TAY Transfer Accumulator to Index Y -- Trasferisci l'accumulatore all'indice Y
TSX Transfer Stack to Index X -- trasferisci lo stack all'indice X
TXA Transfer Index X to Accumulator -- trasferisci l'indice X all'accumulatore
TXS Transfer Index X to Stack Register -- trasferisci l'indice X al registro dello stack
TYA Transfer Index Y to Accumulator -- trasferisci l'indice Y all'accumulatore

APPENDICE VI
SPIEGAZIONE DEI MESSAGGI DI ERRORE

Nel listato dei programmi sono posti alcuni messaggi di errore che accompagnano i comandi errati. Quella che segue è una lista di tutti i messaggi di errore che possono essere prodotti durante la compilazione

MODO NON PERMESSO

Dopo la codifica corretta, separata da uno o più spazi, c'è la lettera A. L'assembler sta tentando di usare l'accumulatore come operando (A indica infatti il modo accumulatore), ma la codifica in quel comando è una di quelle che non permettono di riferirsi all'accumulatore. Controlla di non aver battuto una linea con etichetta A (illegale), al quale quel comando si riferisce. Se stavi provando a riferirti all'accumulatore, controlla gli operandi validi per la codifica usata

A,X,Y,S,P RISERVATI

L'etichetta del comando è uno dei cinque nomi riservati (A,X,Y,S e P). Essi hanno un significato speciale per l'assembler e per questo non possono essere usati come etichette. L'uso di uno di questi nomi fa sì che venga stampato un messaggio di errore e che non venga generato alcun codice per il comando. L'etichetta non viene definita ed appare nella tabella dei simboli come una variabile indefinita. Riferimenti ad una di queste etichette in un qualsiasi punto dèl programma generano messaggi di errore come se l'etichetta non fosse stata affatto dichiarata.

SALTO FUORI LIMITE

Tutte le istruzioni di salto (escluse JMP e JSR) vengono compilate in due byte di codice. Un byte è per la codifica, l'altro indica l'indirizzo a cui saltare. Quest'ultimo è considerato relativamente all'indirizzo del primo byte dell'istruzione successiva a quella di salto. Se il valore del byte è compreso tra 0 e 127, il salto è in avanti; se invece il valore del byte è compreso tra 128 e 255, il salto è all'indietro. Per questo una di queste istruzioni può saltare solamente 127 byte avanti e 128 indietro rispetto all'indirizzo dell'inizio dell'istruzione successiva. Salti al di fuori di questi limiti vengono segnalati con questo messaggio d'errore. Per correggere l'errore devi ristrutturare il programma.

ESPRESSIONE INESATTA

Calcolando un'espressione, l'assembler trova un carattere che non può interpretare come parte di un 'espressione valida. Questo può accadere se il campo che segue la codifica contiene caratteri speciali illegali all'interno di un'espressione (le parentesi, ad es.). Controlla il campo dell'operando ed assicurati che in esso vi siano solamente caratteri validi.

DOPPIA DEFINIZIONE

Il primo campo di una riga non è una codifica ed è interpretato come un'etichetta. Se la linea corrente è la prima in cui quel simbolo compare come etichetta (o se è la prima in cui compare alla sinistra di un segno di uguale), il simbolo stesso viene posto nella tabella dei simboli ed assunto come definito in quella linea.
Al contrario, se il simbolo è già apparso come etichetta oppure alla sinistra di un segno di uguale, in una linea precedente alla linea corrente, l'assembler lo trova già definito nella tabella dei simboli.
Poichè l'assembler non permette la ridefinzione dei simboli, viene stampato questo messaggio di errore.

FILE EXIST
(IL FILE ESISTE GIÀ)

Questo messaggio di errore viene stampato quando su disco è già presente un file con lo stesso nome di quello che si è assegnato al

file oggetto. Si può ovviare a questa situazione cancellando il vecchio file o sostituendo il disco.

FILE NOT FOUND
(FILE NON TROVATO)

Questo messaggio viene stampato quando si presenta uno di questi tre problemi:

non è stato trovato il file sorgente
un comando. LIB specifica un file inesistente
un comando. FIL specifica un file inesistente

L'utente deve assicurarsi di non avere inserito nel drive il disco sbagliato e che il nome del file sia scritto correttamente.

RIFERIMENTO INDIETRO

L'espressione sulla destra di un segno di uguale contiene un simbolo che non è stato definito in precedenza. Una delle operazioni dell'essembler è la valutazione delle espressioni e delle etichette, e l'assegnamento di un valore o di un indirizzo ad esse. L'assembler processa il file sorgente in modo sequenziale, e ciò significa che tutti i simboli incontrati vengono divisi in due classi: quelli già definiti e quelli che vengono trovati per la prima volta, e conseguentemente non sono ancora stati definiti. L'assembler assegna i valori definiti e costruisce una tabella di valori indefiniti. Quando viene scoperto un simbolo usato in precedenza, viene sostituito nella tabella. L'assembler poi processa tutti i comandi una seconda volta usando i valori correntementi definiti.
Un'etichetta o un'espressione che usi un valore non ancora definito è considerata essere riferita più avanti al valore da definire.

Per permettere di valutare correttamente le espressioni, questo assembler permette un livello di riferimento in avanti, cosicché il codice seguente è permesso:

# di linea	etichetta	codifica	operando
100	BNE	NEW	
200	NEW	LDA	# 5

Questo invece non è permesso:

# di linea	etichetta	codifica	operando
100	BNE	NEW	
200	NEW	INC	NEXT + 5
300	NEXT	LDA	# 5

Questa limitazione però non dovrebbe creare troppi problemi nell'uso normale delle etichette; per risolvere il problema dell'esempio è infatti sufficiente fare così:

# di linea	etichetta	codifica	operando
100	BNE	NEW	
200	NEXT	LDA	# 5
300	NEW	INC	NEXT + 5

Questo errore può anche significare che il valore alla destra del segno di uguale non è affatto definito in nessuna parte del programma, nel qual caso la soluzione è uguale a quella per i valori indefiniti

L'assembler non può processare più di un livello di riferimento in avanti.
Tutte le espressioni con simboli alla destra del segno di uguale, per essere valutate correttamente, devono riferirsi esclusivamente a simboli già definiti nelle linee precedenti.

TIPO OPER.
ILLEGALE

Dopo avere trovato una codifica con operando non implicito, l'assembler tratta il campo dell'operando (il campo immediatamente successivo alla codifica che non sia composto di soli spazi) e determina il tipo di operando usato (indicizzato, assoluto, etc.). Se il tipo di operando trovato non è compatibile con la codifica, viene stampato questo messaggio di errore.

Controlla che tipo di operando sono permessi per la codifica in questione ed assicurati che il formato dell'operando sia corretto (vedi la sezione 1.1 sui modi di indirizzamento).

Se il campo di operando inizia con una parantesi, controllalo attentamente. Se è previsto che sia un operando indiretto, ricontrolla il formato corretto per i due tipi disponibili. Se il formato è sbagliato (manca la parentesi di chiusura o il registro indice), viene stampato questo messaggio di errore. In un operando indicizzato controlla anche che i registri in-

OPCODE NON VALIDO

L'assembler cerca nella linea fino a trovare la prima stringa di caratteri diversi dagli spazi. Se la stringa non è una delle 56 codifiche valide, la considera un'etichetta, e la pone nella tabella dei simboli. Poi continua a cercare la stringa successiva. Se non ne trova nessuna, legge la linea successiva e la compilazione prosegue, se invece viene trovato un secondo campo, viene considerato una codifica (poiché è permessa una sola etichetta per riga). Se questa stringa di caratteri non è una codifica valida, viene stampato questo messaggio di errore.

Questo errore può verificarsi quando la codifica è scritta male, nel qual caso l'assembler interpreta la codifica come etichetta (se non c'è altra etichetta). Esso prova poi a compilare il secondo campo come se fosse una codifica. Se esiste un altro campo, viene stampato questo errore.

Ricerca una codifica scritta male o una linea con più di una etichetta.

GLI INDICI SONO X O Y

Dopo avere trovato una codifica valida, l'assembler cerca l'operando. In questo caso il primo carattere nel campo dell'operando è una parentesi. L'assembler interpreta dunque il prossimo campo come un indirizzo indiretto, che, eccetto il comando JUMP, deve essere indicizzato mediante uno dei due registri indice, X e Y. Nel caso in cui il carattere che l'assembler considera essere il registro indice sia diverso da X e da Y, viene stampato questo messaggio di errore.

Se il campo di operando inizia con una parentesi, controllalo attentamente.
Se è previsto che sia un operando indiretto, ricontrolla il formato corretto per i due tipi disponibili.

Se il formato è sbagliato (manca la parentesi di chiusura o il registro indice), viene stampato questo messaggio di errore. In un operando indicizzato controlla anche che i registri indice non manchino o non siano sbagliati (formato: espressione, registro indice).

INDIRET. FUORI LIMITE

L'assembler riconosce un indirizzo indiretto dalle parentesi che lo circondano. Se il campo che segue la codifica è compreso tra parentesi, l'assembler tenta di compilarlo come un indirizzo indiretto. Se il campo dell'operando è maggiore di 255 (necessita quindi due byte per essere espresso), viene stampato questo messaggio di errore.

Questo errore si verifica solo quando il campo dell'operando ha forma corretta (cioè quando un registro indice segue l'indirizzo) e il campo dell'indirizzo è fuori dalla pagina zero. Per correggerlo, il campo dell'indirizzo deve essere riferito alla pagina zero (il byte alto implicito è 00).

INDIRIZZO NON VALIDO

Un indirizzo menzionato in un'istruzione o l'indirizzo di un comando dell'assembler (.BYTE, .DBYTE, .WORD) non è valido. Nel caso di un'istruzione, l'operando generato dall'assembler deve essere maggiore o uguale a zero e minore o uguale a $FFFF (lungo cioè due byte). (Chiaramente si devono escludere gli indirizzi delle istruzioni di salto relativo, che devono essere compresi tra -128 e +127 byte dall'istruzione successiva). Se l'operando genera più di due byte di codice, oppure se è minore di zero, viene stampato questo messaggio di errore.

Per il comando .BYTE, ogni operando è limitato ad un byte. Tutti i riferimenti di indirizzo devono essere maggiori o uguali a zero.

La validità è controllata dopo che l'operando è stato calcolato. Controlla i valori dei simboli usati nel campo dell'operando (vedi tabella dei simboli).

INIZIO LABEL CON A-Z

Il primo campo che non sia composto di soli spazi non è una codifica valida. Per questo l'assembler prova ad interpretarlo come un'etichetta, ma il primo carattere non è alfabetico. Viene stampato questo messaggio di errore.

Cerca un comando composto dal solo campo dell'operando che inizia con un carattere speciale. Cerca anche linee la cui etichetta sia scritta male.

LABEL TROPPO LUNGA

Tutti i simboli sono limitati a sei caratteri di lunghezza. Mentre compila, l'assembler cerca uno dei caratteri separatori (di solito uno spazio) per individuare la fine di una stringa. Se è stato usato un carattere illegale come separatore, viene stampato il messaggio di errore, in quanto il falso carattere separatore fa sì che la lunghezza della stringa ecceda i sei caratteri. Cerca i caratteri diversi dallo spazio tra etichette e codifiche. Cerca anche linee di commento che inizino con una parola più lunga di sei caratteri e non preceduta da un punto e virgola. In questo caso l'assembler sta provando ad interpretare parte del commento come etichetta.

NON - 'ALFANUMERICO

Le etichette sono composte da sei caratteri alfanumerici. Il campo dell'etichetta deve essere separato dal campo della codifica da uno o più spazi. Se c'è un carattere speciale o un altro carattere fra l'etichetta e la codifica potrebbe essere stampato questo messaggio di errore.

Ognuna delle 56 codifiche valide è composta da tre caratteri alfabetici. Esse devono essere separate dal campo dell'operando (sempre che ce ne sia uno) da uno o più spazi. Se la codifica finisce con un carattere speciale (come una virgola), viene stampato questo messaggio di errore.

Un'etichetta da sola o una codifica che non abbia bisogno di operando possono essere seguite direttamente da un punto e virgola, per indicare che il resto della linea è un commento (l'uso di un punto e virgola sposta il commento al punto di tabulazione successivo).

PC NEGATIVO -- RESET 0

Un programma assemblato è caricato nella RAM nell'intervallo di indirizzi tra 0 e 64K (65535). Questi sono i limiti del computer. Per definire un indirizzo possono essere usati al massimo due byte. Poiché non esiste memoria negativa, il tentativo di riferirsi ad un indirizzo negativo causa questo errore, ed il contatore di programma (o il puntatore della locazione di memoria corrente) viene posto a zero.

Quando questo accade, l'assembler continua ad assemblare il codice con il nuovo valore del contatore di programma. Questo può far sì che più byte vengano assemblati nelle medesime locazioni. Per questo, stai molto attento a mantenere il contatore di programma nei limiti consentiti.

CAMPO NON COMPLETO

Questo messaggio di errore viene stampato quando l'assembler sta cercando un campo necessario ed incontra la fine della linea corrente prima di trovare il campo stesso. Bisogna ricercare le condizioni seguenti: un campo di codifica valido senza il campo di operando sulla stessa riga; una codifica che si pensava potesse agire con operando implicito, e che invece necessita di un operando; una stringa ASCII a cui manca la virgoletta di chiusura (assicurati che tutte le virgolette comprese nella stringa siano raddoppiate: per ottenere una virgoletta alla fine di una stringa ne devi inserire tre, per una in mezzo ne devi inserire due, per chiudere la stringa solamente una); una virgola alla fine del campo dell'operando indica che devono esserci altri operandi: se non ci sono altri operandi, l'assembler oltrepassa la fine della linea e segnala l'errore.

READ ERROR (ERRORE DI LETTURA)

Questo messaggio comunica un errore di lettura da parte del disco. Per una descrizione degli errori e delle loro cause, guarda il manuale del disk drive.

DIRETTIVA INDEFINITA

Tutti i comandi riservati all'assembler (.BYTE, .WORD, etc) cominciano con un punto. Se il primo carattere di un campo è un punto, l'assembler interpreta i caratteri successivi come uno di questi comandi.

Se però la stringa di caratteri non è un co-

mando valido, viene stampato questo messaggio di errore.

Cerca i comandi scritti male ed i punti all'inizio di campi che non contengono comandi (campo dell'etichetta, della codifica, dell'operando).

SIMBOLO NON DEFINITO

Questo errore è generato dal secondo passaggio dell'assembler. Se nel primo passaggio l'assembler trova un simbolo nel campo degli operandi (il campo che segue la codifica o che si trova a destra di un segno di uguale) che non è ancora stato definito, l'assembler pone il simbolo nella tabella perché possa essere interpretato nella seconda passata. Se il simbolo è definito (ovvero se sta a sinistra di un segno di uguale o se è il primo campo in una linea), la prima passata determina il suo valore e lo inserisce nella tabella dei simboli. Per questo un simbolo in un campo di operando, trovato prima di essere stato definito, viene definito durante la seconda passata. In questo caso la compilazione può essere completata.

Se invece la prima passata non trova il simbolo come etichetta o posto a sinistra di un segno di uguale, l'assembler non lo inserisce nella tabella dei simboli come un simbolo definito. Quando la seconda passata prova ad interpretare il campo dell'operando in cui si trova quel simbolo, non c'é un valore corrispondente al simbolo, ed il campo non può essere interpretato. Per questo motivo viene stampato questo messaggio di errore.

Questo errore si verifica anche se uno dei nomi riservati A,X,Y,S o P viene usato come etichetta e ricorre in qualche parte del programma.
Nel comando che contiene il simbolo riservato, l'assembler lo vede come un simbolo indefinito. Per correggere questi errori controlla di non aver usato uno dei simboli riservati, di non aver scritto male delle etichette o di non averne dimenticata qualcuna.

Nota: Quando l'assembler trova un'espressione (in un campo di operando o a destra di un segno di uguale), esso prova a calcolarne il valore. Se in essa ci sono simboli non ancora definiti l'assembler la segnala come un riferimento in avanti ed aspetta a calcolarla nella seconda passata. Se l'espressione si trova sulla parte destra di un segno di uguale, il riferimento in avanti è un errore e viene segnalato come tale. Invece, se l'espressione è nel campo dell'operando di una codifica valida, la prima passata riserverà due byte per il valore dell'espressione e la segnala come un riferimento in avanti. Quando la seconda passata riempe il valore dell'espressione ed il valore è lungo un solo byte, l'istruzione risulta di un byte più lunga del necessario. Questo è il motivo per cui i riferimenti in avanti alla pagina zero sprecano un byte di memoria (quello in più che era stato riservato). Durante la prima passata l'assembler non sapeva quanto fosse grande il valore, così ha riservato memoria per il valore più grande, lungo due byte.

APPENDICE VII
Sommario dei comandi dell'Editor 64

Comando	Descrizione
AUTO n. 1	Comincia la numerazione automatica
AUTO	Pone fine alla num. automatica
CHANGE/ s1/s2/, n1-n2	Cambia una stringa in un intervallo
CHANGE/ s1/s2/	Cambia una stringa in tutto il file
CPUT "FILE"	PUT compattato, gli spazi inutili vengono rimossi
DELETE n1-n2	Cancella un intervallo di linee
FIND/ s1/,n1-n2	Cerca una stringa in un intervallo di linee
FIND/s1/	Cerca una stringa in tutto il file
FORMAT n1-n2	Stampa formattata
GET "FILE", n1,n2,n3	Carica il testo da un file su disco
GET "FILE"	Forma semplificata di GET
KILL	Disabilita l'editor
LIST	Lista linee di testo
NUMBER n1,n2,n3	Rinumera il testo
PUT "FILE", n1,n2,n3,n4	Salva il testo su un file su disco

PUT "FILE"	Forma semplificata di PUT

APPENDICE VIII
Sommario dei comandi del Monitor

A - ASSEMBLE	Assembla una linea di codice
C - COMPARE	Confronta due zone di memoria e stampa tutti i byte che le differenziano
D - DISASSEMBLE	Disassemblea una linea di codice del 6502
F - FILL	Riempe la memoria con il valore specificato
G - GO	Lancia l'esecuzione all'ind. specificato
H - HUNT	Cerca nella memoria le ricorrenze di alcuni byte
I - INTERROGATE	Mostra i valori ASCII delle locazioni di memoria desiderate
L - LOAD	Carica un file da cassetta o disco
M - MEMORY	Mostra i valori esadecimali delle locazioni di memoria desiderate
N - NEW LOCATOR	Riloca i programmi in codice macchina dopo che sono stati spostati
R-REGISTERS	Mostra i registri della CPU
S-SAVE	Salva su disco o cassetta
T-TRANSFER	Sposta del codice da una zona di memoria all'altra
X-EXIT	Esce dal MON64 (bisogna resettare il BASIC)

APPENDICE IX
Sommario dei comandi del Dos Wedge 64

Comando	**Descrizione**
@	Stato corrente del disco
@ C(dr): nfile ([vol]) = vfile ([vol])	Copia un file
@I (dr)	Inizializza un drive
@N (dr): nomedisco, id	Formatta un disco
@Q	Esce dal programma WEDGE64
@R(dr): nfile ([vol]) = vfile ([vol])	Cambia nome ad un file
@S(dr): nfile	Cancella un file
@UJ	Resetta il DOS
@$(dr):(nfile) (*) ([vol])	Legge la directory
#n	Tutti i comandi DOS andranno a n, ove n è compreso tra 8 e 15 inclusi.
/nomefile	Carica un file (all'indirizzo del BASIC)
%nomefile	Carica un file (al suo indirizzo di caricamento)
↑nomefile	Carica un file (all'indirizzo BASIC e lo lancia)
←nomefile	Salva un file

Nota bene:
'vol' è qualsiasi carattere racchiuso tra parentesi quadre: 'dr' deve essere 0 o 1 per i rispettivi drives.

QUESTO MESE IN EDICOLA

CHI CAPISCE DI COMPUTER SA CHE QUESTE SONO LE RIVISTE E LE CASSETTE CHE NON TRADIRANNO MAI.

NON PERDETEVELE!!!

E ATTENTI ALLA DATA DI USCITA...



I MAGNIFICI SETTE

SPECIAL PLAYGAMES

PER IL CBM 64 E LO SPECTRUM 48 K

N. 26 - in edicola il 2 febbraio